BLUE: a software package to combine correlated estimates of physics observables within ROOT using the Best Linear Unbiased Estimate method — Program manual

Release 2.4.0

Richard Nisius

March 10, 2020

Max-Planck-Institut für Physik (Werner-Heisenberg-Institut)
Föhringer Ring 6, D-80805 München, Germany,
http://www.mpp.mpg.de/˜nisius,
Richard.Nisius@mpp.mpg.de

Abstract

The combination of correlated estimates of a number of observables is a common task in particle physics. This is frequently performed using the Best Linear Unbiased Estimate (BLUE) method. Given the widespread usage of the ROOT analysis package, a flexible ROOT implementation of the BLUE mathematical framework was written, and is described in this manual. The software is freely available from the corresponding `hepforge` project page. Given it is based on ROOT, it is distributed under the GNU Lesser Public License.

Contents

1 Introduction

The combination of a number of estimates for a single observable is discussed in Ref. [1]. Here, the term estimate denotes a particular outcome (measurement) of an experiment based on an experimental estimator (an algorithm for a measurement) of the observable, which follows a probability density function (pdf). The particular estimate obtained by the experiment may be a likely or unlikely outcome for that pdf. Repeating the measurement numerous times under identical conditions, the estimates will follow the underlying pdf of the estimator. The analysis makes use of a χ^2 minimisation to obtain the combined value. In Ref. [1], this minimisation is expressed using the mathematically equivalent BLUE ansatz.

Provided the estimators are unbiased, when applying this formalism the **Best Linear Unbiased Estimate** of the observable is obtained with the following meaning: **Best:** the combined result for the observable obtained this way has the smallest variance; **Linear:** the result is constructed as a linear combination of the individual estimates; **Unbiased Estimate:** when the procedure is repeated for a large number of cases consistent with the underlying multi-dimensional pdf, the mean of all combined results equals the true value of the observable. The extension to more than one observable is described in Ref. [2].

For many years, a freely available Fortran based software [3] to perform the combination for a number of estimates and for several observables was widely used. The implementation of the BLUE method described here is integrated into the ROOT analysis framework [4]. The equations to solve the problem for the general case of m estimates of n observables with $m \geq n$ can be found in Ref. [2]. They are implemented in the software presented, but are not repeated here. However, the simple case of two correlated estimates of the same observable is discussed in some detail. This is because already for this case the main features of the combination can easily be understood. For further information and the derivation of the formulas listed below see Ref. [5].

Let x_1 and x_2 with variances σ_1^2 and σ_2^2 be two estimates from two unbiased estimators X_1 and X_2 of the true value x_T of the observable, and ρ the total correlation of the two estimators. For Gaussian uncertainties the two-dimensional estimator pdf reads:

$$P(X_1, X_2) = \frac{1}{\sqrt{2\pi}\sigma_1}\frac{1}{\sqrt{2\pi}\sigma_2}\frac{1}{\sqrt{1-\rho^2}} \cdot \\ \exp\left\{-\frac{1}{2(1-\rho^2)}\left(\frac{(X_1-x_\mathrm{T})^2}{\sigma_1^2} + \frac{(X_2-x_\mathrm{T})^2}{\sigma_2^2} - \frac{2\rho(X_1-x_\mathrm{T})(X_2-x_\mathrm{T})}{\sigma_1\sigma_2}\right)\right\} \quad (1)$$

Without loss of generality it is assumed that the estimate x_1 stems from an estimator X_1 of x_T that is at least as precise as the estimator X_2 yielding the estimate x_2, such that $z \equiv \sigma_2/\sigma_1 \geq 1$. In this situation the BLUE x of x_T is:

$$x = (1-\beta)\,x_1 + \beta\,x_2\,,$$

where β is the weight of the less precise estimate, and, by construction, the sum of weights is unity. The variable x is the combined result and σ_x^2 is its variance, i.e. the uncertainty assigned to the combined value is σ_x. To investigate the improvement on the precision of x when adding the information of x_2 to the more precise estimate from x_1, i.e. to decide whether it is worth combining, the variable σ_x/σ_1 is investigated. This variable quantifies the uncertainty in the combined value in units of the uncertainty in the more precise estimate, i.e. $1 - \sigma_x/\sigma_1$ is the relative improvement achieved by also using x_2 from the less precise estimator.

The two quantities and their derivatives with respect to the parameters ρ and z are given in Eqs. 2–7, see Ref. [5]. They are valid for $-1 \leq \rho \leq 1$ and $z \geq 1$, except for $\rho = z = 1$. The resulting variations of the combined value are given in Eqs. 8–9.

$$\beta = \frac{x - x_1}{x_2 - x_1} = \frac{1 - \rho z}{1 - 2\rho z + z^2} = \frac{1 - \rho z}{(1 - \rho z)^2 + z^2(1 - \rho^2)} \tag{2}$$

$$\frac{\sigma_x}{\sigma_1} = \sqrt{\frac{z^2(1 - \rho^2)}{1 - 2\rho z + z^2}} \tag{3}$$

$$\frac{\partial \beta}{\partial \rho} = \frac{z(1 - z^2)}{(1 - 2\rho z + z^2)^2} \tag{4}$$

$$\frac{\partial \frac{\sigma_x}{\sigma_1}}{\partial \rho} = z(z - \rho)(1 - \rho z)\sqrt{\frac{1}{(1 - \rho^2)(1 - 2\rho z + z^2)^3}} \tag{5}$$

$$\frac{\partial \beta}{\partial z} = \frac{\rho(1 + z^2) - 2z}{(1 - 2\rho z + z^2)^2} \tag{6}$$

$$\frac{\partial \frac{\sigma_x}{\sigma_1}}{\partial z} = (1 - \rho z)\sqrt{\frac{1 - \rho^2}{(1 - 2\rho z + z^2)^3}} \tag{7}$$

$$\frac{\partial x}{\partial \rho} = (x_2 - x_1)\frac{\partial \beta}{\partial \rho} \tag{8}$$

$$\frac{\partial x}{\partial z} = (x_2 - x_1)\frac{\partial \beta}{\partial z} \tag{9}$$

The resulting β and σ_x/σ_1, as functions of ρ, and for various z values (Eq. 2 and Eq. 3) are shown in Figures 2a and 2b. A few features of the variables β and σ_x/σ_1 discussed below are important to understand the results of the combination.

The value of β is smaller or equal to 0.5, because otherwise x_2 would be the more precise estimate. Since the denominator in Eq. 2 is positive for all allowed values of ρ and z, the function for β turns negative for $\rho > 1/z$ as shown in Figure 2a. As can be seen from the second term in Eq. 2, the value of β can be interpreted as the distance of the combined value from the more precise estimate in units of the difference in the two estimates. When β is negative, the signs of the numerator and denominator are different. This means the value of x lies on the opposite side of x_1 than x_2 does, or in other words, the combined value lies outside the range spanned by the two estimates.

Since the denominator in Eq. 2 and Eq. 3 are identical, and the denominator of Eq. 2 equals the numerator of Eq. 3 plus an additional term that is positive for all values of ρ and z, the value of σ_x/σ_1 is always smaller than 1 as shown in Figure 2b. Again this is expected, since including the information from the estimate x_2 should improve on the knowledge of x, which means on its precision σ_x. Not surprisingly, the value of σ_x/σ_1 is exactly one for $\rho = 1/z$, i.e. when $\beta = 0$. In this situation, the information from x_2 is ignored in the linear combination, and consequently $x = x_1$ and $\sigma_x = \sigma_1$.

The derivatives of β and σ_x/σ_1 with respect to ρ as functions of ρ, and for various z values (Eq. 4 and Eq. 5) are shown in Figures 2c and 2d. The equations for β and σ_x/σ_1, this time as a function of z and for various ρ values, are shown in Figures 3a and 3b. Finally, the derivatives of β and σ_x/σ_1 with respect to z as functions of z, and for various ρ values (Eq. 6 and Eq. 7) are shown in Figures 3c and 3d. These derivatives can be used to evaluate the sensitivity of the combined result to the imperfect knowledge on both the correlation ρ and the uncertainty ratio z of the individual estimators. With this information the stability of the combined result can be assessed and a decision can be taken on whether to refrain from combining. This decision should only be based on the parameters of the combination but not on the outcome for a particular pair of estimates x_1 and x_2. This is because these parameters are features of the underlying two-dimensional pdf of the estimators, whereas the two specific values are just a pair of estimates, i.e. a single possible likely or unlikely outcome of results.

This manual is organised as follows: The software structure is outlined in Section 2, followed by the description of the user interface given in Section 3. The examples provided are discussed in Section 4. The conversion of input files for the Fortran software [3] to functions to be used with this ROOT implementation is explained in Section 5. Some hints on the installation and usage of the software are given in Section 6. Conclusions are drawn in Section 7, followed by a list of recent changes made to the software given in Appendix A.

2 Software structure

This section explains the general strategy for the usage of the package. The details of the functions mentioned here are given in Section 3. The functionality is implemented in a ROOT class called BLUE that derives from `TObject`. No attempt was made to override the default implementation provided by this, except for what is described below. The usage of the software is schematically shown in Figure 1. It is separated into a number of steps.

1. Create: During this first step, the constructor is called, thereby defining the number of estimates, uncertainties and observables.
2. Fill: The individual estimates and their uncertainties, as well as all estimator correlation matrices for the uncertainty sources are filled. Optionally, also the statistical precision in the uncertainties and names for estimates, uncertainty sources and observables can be filled. When the mandatory input is completed, the input stream is closed automatically and all filling functions are disabled.
3. Change: In this (optional) step, individual estimates and/or uncertainty sources can be disabled, or correlation assumptions can be altered for the combination to follow by calling the corresponding `Set...()` functions.
4. Fix: Before a combination can be performed, the inputs to the combination have to be fixed by calling `FixInp()` indicating the end of the selection. After this call, several `Print...()` functions are available for digesting the inputs and the selections made. In addition, `Get...()` functions are available to retrieve information on the selected inputs to the next combination.
5. Combine: In this step the actual combination is performed by calling one of the `Solve...()` functions.
6. Digest: At this point, additional `Print...()` functions are enabled for digesting the result

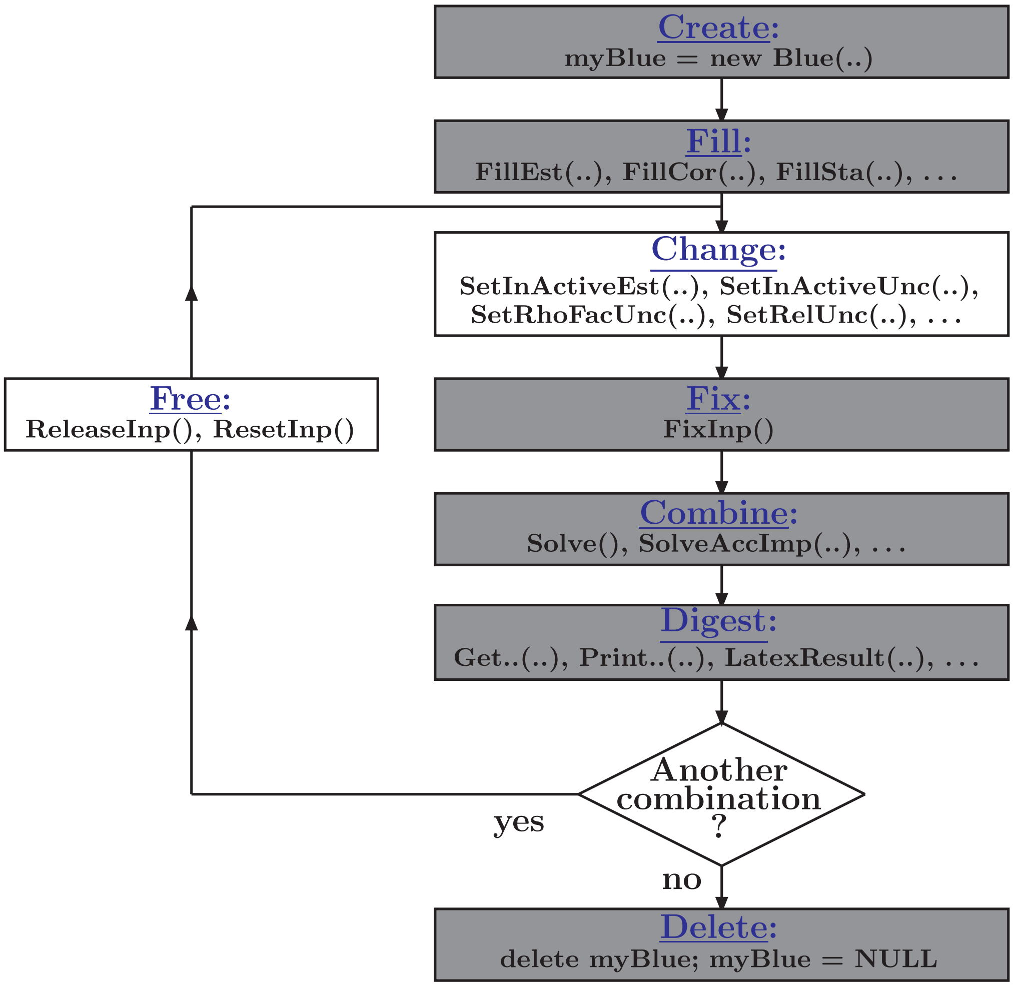

Create:
myBlue = new Blue(..)
Fill:
FillEst(..), FillCor(..), FillSta(..), . . .
Change:
SetInActiveEst(..), SetInActiveUnc(..),
SetRhoFacUnc(..), SetRelUnc(..), . . .
Free:
ReleaseInp(), ResetInp()
Fix:
FixInp()
Combine:
Solve(), SolveAccImp(..), . . .
Digest:
Get..(..), Print..(..), LatexResult(..), . . .
Another combination ?
yes
no
Delete:
delete myBlue; myBlue = NULL

Figure 1: The work flow for combining with the BLUE software. The gray boxes display the steps to be performed for a single combination with unchanged inputs. The white boxes display the optional steps for further combinations with changed inputs and/or different solving methods. In each rectangular box, the first line indicates the performed action, while the second line lists the functions to use.

for the observables. In addition, `Get...()` and `Display...()` functions are available to analyse the result of the combination.

7. Free: The steps 3-6 can be performed as often as wanted. In this case, after any combination, first the input has to be freed for further selections by calling either `ReleaseInp()` or `ResetInp()`. The difference of these two options is discussed below.
8. Delete: If no further combination is wanted, the object is deleted.

3 Details of the interface

This section describes the details of the interface. All arguments passed to member functions are declared as `const`, except for those that are return values as described below. However, this fact is not mentioned in the description of the function prototypes. This means arguments denoted as `Int_t` in fact are `const Int_t`. In contrast, functions that are `const`, i.e. those that do not alter the state of the object, are marked as such.

3.1 Constructor

`Blue(Int_t NumEst, Int_t NumUnc, Int_t NumObs, Int_t* IWhichObs, Int_t* IWhichFac)`:
`Blue(Int_t NumEst, Int_t NumUnc, Int_t NumObs, Int_t* IWhichObs)`:
`Blue(Int_t NumEst, Int_t NumUnc, Int_t* IWhichFac)`:
`Blue(Int_t NumEst, Int_t NumUnc)`:
The first constructor instantiates the object for a number of estimates (`NumEst`), uncertainty sources (`NumUnc`) and observables (`NumObs`). The array `IWhichObs` indicates which observable a given estimate is determining. The array `IWhichFac` defines different groups to be considered in systematic variations of the correlation assumptions, when using `SolveScaRho()`, see below. The input for the example of four estimates, ten uncertainty sources for two observables, where the first two estimates determine the first observable, and the second two estimates determine the second observable is: `NumEst = 4`, `NumUnc = 10`, `NumObs = 2`, and `IWhichObs = {0,0,1,1}`. If these fall into two groups of estimates, e.g. (0, 2) and (1, 3), which e.g. could stem from different experiments, and for which the correlation assumption should be scanned differently for the pairs of estimates from the same experiment (0, 2) and (1, 3), or from different experiments (0, 1), (0, 3), (2, 1) and (2, 3), the following info should be provided:

$$\texttt{IWhichFac} = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}, \tag{10}$$

where the array `IWhichFac` should contain this matrix in row wise storage. The values on the diagonal are not relevant, the off-diagonal elements should start from zero and run up to ℓ `= NumFac-1`, where `NumFac` is the number of groups desired.

In the case of a single observable, i.e. if `NumObs = 1`, the information in `IWhichObs` is redundant and ignored. In this case the more simple constructors can be used instead. If also possible scans in `SolveScaRho()` should be performed simultaneously for all pairs of estimates, the last constructor is sufficient.

3.2 Fill input

The software has mandatory and optional input which are discussed in turn.

3.2.1 Mandatory input

`void FillEst(TVectorD* v, TMatrixD* e):`
`void FillEst(TMatrixD* v, TMatrixD* e):`
`void FillEst(TMatrixD* x):`
`void FillEst(Double_t* x):`
`void FillEst(Int_t` i`, Double_t* x):`
Using the last implementation, the estimate i with the index in the following range: i `= 0, ...,` `NumEst-1` is filled. The array `x` must contain `NumUnc + 1` entries, the value of the estimate and the individual uncertainties in the following form: `x` $= \{$`Value`$, \sigma_0, \sigma_1, \ldots, \sigma_{k_{\max}}\}$ with $k_{\max}$ `= NumUnc - 1`. The software assumes that σ_0 is the statistical uncertainty and σ_k with $k > 0$ are systematic uncertainties. If for a source k a negative entry $\sigma_k < 0$ is supplied, this value is considered a percentage uncertainty. During filling this is converted from $\sigma_k \to -\sigma_k \cdot$ `Value / 100`.

The other implementations allow for a more compact data structure. Here, `v` should contain the values of the estimates either as a `TVectorD` of dimension `v(NumEst)` or as a `TMatrixD` of dimension `v(NumEst,1)`. The structure `e` should contain the uncertainties in the estimates as a `TMatrixD` of dimension `e(NumEst,NumUnc)`. Finally, the structure `x` should contain all estimates and their uncertainties, i.e. the `TMatrixD` should have dimension `x(NumEst,NumUnc+1)`, and the array of `Double_t` values the dimension `x(NumEst*(NumUnc+1))`.

`void FillCor(TMatrixD* x[]):`
`void FillCor(Int_t` k`, TMatrixD* x):`
`void FillCor(Int_t` k`, Double_t* x):`
Using the last implementation, the matrix of the estimator correlations for the uncertainty k with indices in the range $k = 0, \ldots,$ `NumUnc-1` is filled. For the example of `NumEst = 3` the matrix of estimator correlations for any uncertainty source k is:

$$V = \begin{pmatrix} V_{00} & V_{01} & V_{02} \\ V_{10} & V_{11} & V_{12} \\ V_{20} & V_{21} & V_{22} \end{pmatrix} . \tag{11}$$

The array `x` must contain the row wise storage of this matrix, i.e. for the above example it should read `x` $= \{V_{00}, V_{01}, V_{02}, V_{10}, V_{11}, V_{12}, V_{20}, V_{21}, V_{22}\}$. The user should ensure the matrix to be a valid correlation matrix, i.e. the elements to be within bounds, the matrix to be symmetric, and the diagonal elements to be unity, i.e. the following conditions should be fulfilled: $V_{ii} = 1$ and $-1 \leq V_{ji} = V_{ij} \leq 1$ for $i \neq j$, for all $i, j = 0, \ldots$ `NumEst - 1`. If the matrix is not symmetric, or off diagonal elements are outside their range of validity, the input is not consistent. In this case, an error message is issued and the software will refrain from combining. In any case, the diagonal elements will be forced to unity by the software.

Given the above relations, the entire information is contained in one half of the off diagonal elements (e.g. those marked in red in Eq. 11). To account for this, this function can also be called with k replaced by $-k$ (for $k \neq 0$). In this case the array `x` should only contain the significant elements again in row-wise storage, i.e. in the above case `x` $= \{V_{01}, V_{02}, V_{12}\}$ is expected by the software. Again, if elements are outside their range of validity, the input is not consistent, an error message is issued and the software will refrain from combining.

The other two implementations allow for a more compact data structure. The structure `x` should contain the values of the correlations of the estimates for a given source of uncertainty k as a `TMatrixD` of dimension `x(NumEst,NumEst)`. The first implementation uses an array of pointers to all `NumUnc` matrices. In the second implementation for each source k a pointer to the corresponding matrix is passed. This implementation can also be called using $-k$ as explained above.

`void FillCor(Int_t` k`, Double_t rho)`:
Frequently, for some uncertainty sources the estimators are assumed to be either uncorrelated or fully correlated. In this case, only a single value, namely the overall correlation obeying $-1 \leq \texttt{rho} = \rho_k \leq 1$ is significant. A call to this function will store a correlation matrix with $V_{ii} = 1$ and $V_{ji} = V_{ij} = \rho_k$ for $i \neq j$, for $i, j = 0, \ldots,$ `NumEst - 1` for the source k. If the value of ρ_k is not within bounds, the input is not consistent, an error message is issued and the software will refrain from combining.

3.2.2 Optional input

The following input is optional, and its presence is not verified in the automatic recognition of the end of input of estimates and correlations. Since the functions can only be called before the end of input, it is recommended to fill this information before filling estimates and correlations.

`void FillSta(TMatrixD* x)`:
`void FillSta(Double_t* x)`:
`void FillSta(Int_t` i`, Double_t* x)`:
These methods allow to assign a statistical precision to each uncertainty in all estimates, see Ref. [6] for an explanation of this concept. This information will be exploited in the stability test of the solving method `SolveScaSta()` explained below. This solving method requires all information to be filled. Zero values are accepted and will lead to no variation. The structure `x` should contain the values of the statistical precision for all sources of uncertainty and all estimates. The implementation is analogous to the one of the `FillEst()` methods. Please consult their description for further details. The dimension of the `TMatrixD` is: `x(NumEst,NumUnc)`.

The following functions allow to assign names to estimates, uncertainties and observables. They are implemented as `TString` objects. The length of each name is arbitrary, however all printing functions and display routines are optimised for names with equal length of seven characters. The type of characters can be freely chosen. For all functions it is the responsibility of the user to ensure the correct length of the arrays of names, i.e. names for `NumEst` estimates, `NumUnc` uncertainties and `NumObs` observables should be provided.

`void FillNamEst(TString* NamEst)`:
A call to this function will store the names of the estimates.

`void FillNamUnc(TString* NamUnc)`:
A call to this function will store the names of the uncertainties.

`void FillNamObs(TString* NamObs):`
A call to this function will store the names of the observables.

3.3 Fix and free input

`void FixInp():`
The input is fixed for solving and the calculation of several matrices is initiated.

`void ReleaseInp():`
The input is freed for additional selections. Any further selection starts from the situation at the last call to `FixInp()`.

`void ResetInp():`
The input is freed for additional selections. However, in this case any further selection starts from the original user input.

3.4 Solver

The default method for solving the problem is:

`void Solve():`
The BLUE combination for the presently active estimates and uncertainties is performed.

In the following a number of specific `Solve...()` functions are discussed which themselves call `FixInp()` and `Solve()` several times. As a consequence, after calling one of these functions the output of the print functions related to estimates and uncertainties in most cases will be different from the one after the last user call to `FixInp()`. In contrast, since these functions use `ReleaseInp()`, the situation in terms of active estimates, uncertainties and correlation assumptions remains unchanged. Exceptions are: `SolvePosWei()`, where estimates resulting in negative weights are disabled at return, and `SolveMaxVar()`, where the correlations of the estimates for various uncertainty sources are scaled, see `SetRhoFacUnc()` for details. For the user to get to a clean situation after using these exceptions it is recommended to use `ResetInp()` before subsequent selections.

`void SolveRelUnc(Double_t Dx):`
The BLUE combination is performed for the presently active estimates and uncertainties, of which at least one has to be a Rel-ative Unc-ertainty. Iterations are made until, for all active observables, the relative difference in the combined value with respect to the one from the previous iteration falls below `Dx` percent, or until two hundred iterations are reached.

The uncertainty sources can be an arbitrary mixture of relative or absolute uncertainties, see `SetRelUnc(...)` for how to steer this. The term absolute uncertainty means that the value of the uncertainty is identical for all possible values of the estimator pdf, i.e. it is independent of the actual value of the estimate. This means it is the same for the actual estimate, any combined value and the true value. Therefore, irrespectively whether it was calculated for the estimate it also applies to the combined value. In contrast, a relative uncertainty (e.g. of some percent) depends on the actual value of x_{T}. Therefore, for relative uncertainties, the uncertainty assigned to the estimate, $\sigma_i = \sigma_i(x_{\mathrm{i}})$, is formally incorrect, since it should correspond to the uncertainty in the estimator pdf, i.e. $\sigma_i = \sigma_i(x_{\mathrm{T}})$, which has a different value. This means that, in the presence of relative uncertainties, the BLUE method is only an approximation. In this approximation, after each iteration the uncertainty is replaced by the expected uncertainty in the true value x_{T}, approximated by the one of the combined value x. In general, this is a good approximation, see Ref. [5] for a detailed discussion and a number of examples. A utility is provided to compare this to the result obtained from a simplified maximum likelihood approach, see `InspectLike()`.

The procedure of this solver is implemented as follows: First a BLUE combination is performed. Then the uncertainties are adjusted based on the result and the next iteration is performed. This is repeated until convergence is reached. For each estimate i and each relative uncertainty k the dependence of the contribution from this source to the covariance matrix can be defined by the user as a second order polynomial in x. The function reads $\sigma_{ik}^2 = a_0 + a_1\,|x| + a_2\,x^2$. See `SetRelUnc(...)` for the details of the implementation and Ref. [7] for an example of a non linear situation.

`void SolveAccImp(Int_t ImpFla, Double_t Dx):`
`void SolveAccImp(Double_t Dx):`
For each observable a combination of the estimates is performed Acc-ording to their Imp-ortance. For the first implementation, three definitions of importance of the estimates j are implemented given the most precise estimate is i. The second uses the recommended default, i.e. `ImpFla = 0`. The following options are implemented:
`ImpFla = 0` means sorted by decreasing $1 - \sigma_x/\sigma_1$ calculated from Eq. 3 using $12 = ij$
`ImpFla = 1` means sorted by decreasing absolute BLUE weights $|\alpha_j|$
`ImpFla = 2` means sorted by decreasing inverse variance $1/\sigma_j^2$.

The options differ in the estimator correlations that are taken into account. The first accounts for the correlation of the pair of estimates, the second for those of all estimates and the third completely ignores all correlations.

The software suggests which estimates to combine until the uncertainty in the combined value is never improved by more than `Dx` percent by adding further estimates. First a BLUE combination for the presently active estimates and uncertainties is performed. For each active observable the related estimates are sorted by importance. According to this list one estimate at a time is added to the most precise one and the combination is performed, while all less important estimates of this observable are disabled. In contrast, all estimates of other observables are kept active such that the full correlation is preserved. This is repeated for all active observables. The outcome can be digested by a call to `PrintAccImp()`.

`void SolveScaRho(Int_t RhoFla, Double_t* MinRho, Double_t* MaxRho):`
`void SolveScaRho(Int_t RhoFla):`
`void SolveScaRho():`
This function performs a scan in the correlation assumptions for all active estimates, uncertainty sources k, and observables, while using `NumFac` groups (see the constructor) of multiplicative factors r, performing ten steps each in the range defined by `MaxRho` $> r >$ `MinRho`, while decreasing r. For non of the active uncertainties the estimator correlations are allowed to be

declared as `changed` or `reduced`, see `SetRho...Unc()` below for the definitions. While the groups ℓ are always scanned independently, the sources k are scanned either independently for `RhoFla = 0`, or simultaneously for `RhoFla = 1`.

Given that the sources of uncertainty k in general are uncorrelated, because otherwise quadratically adding their contributions to calculate the total uncertainty would not be correct, an independent scan, i.e. `RhoFla = 0` is recommended. See Ref. [5] for a detailed discussion. If this is wanted, and the variation for all sources and groups (k, ℓ) should be done in the range $1 > r > 0$ with respect to the initially provided correlation, the last implementation should be used. Otherwise the boundaries should be given in the following form: `MinRho(`k`=0` ℓ`=0,` k`=0` ℓ`=1, ...,` k`=NumUnc-1` ℓ`=NumFac-1)`.

Manipulations with many groups ℓ that may end up in manipulating single entries of the covariance matrix, can easily lead to instable matrix inversions. The software is protected against this.

The implementation proceeds as follows. First a BLUE combination is performed for the active estimates and uncertainties treating all correlations as scaled correlations, while using any given scale provided by preceding calls to `SetRhoFacUnc()`, and for $r = 1$. Then a scan is performed and the differences in the observables and their uncertainties with respect to the values from the initial result are stored. Finally, the outcome can be digested by a call to `PrintScaRho()`, where inversion failures are indicated by values of `-1.00` for both differences.

`void SolveInfWei()`:
This function is only available for a single observable. It yields the same result as a call to `Solve()` but also calculates the information weights defined in Ref. [8]. The weights calculated are: the BLUE weights α_j, the intrinsic weights, i.e. the inverse variances scaled by the variance of the combined result, the weight assigned to the correlation, the marginal weights and finally the relative weights. These weights are defined as follows:

$$\begin{aligned}
BLUE &= \alpha_j \\
intrinsic &= \frac{\sigma_x^2}{\sigma_j^2} \\
correlation &= 1 - \Sigma_j\, intrinsic \\
marginal &= 1 - \frac{\sigma_x^2}{\sigma_{x,m-j}^2} \\
relative &= \frac{|\alpha_j|}{\Sigma_j |\alpha_j|}
\end{aligned}$$

Here $\sigma_{x,m-j}^2$ denotes the variance of the combination when using all m estimates, except the estimate j. The outcome can be digested by a call to `PrintInfWei()`. *NOTE:* Given the reduction of the combined uncertainty at both sides of the maximum of Eq. 3, see Figure 2b, *absolute* weights are useful for ranking the importance of measurements for the combination. However, the probabilistic interpretation of *relative* weights has to be made with care, see Ref. [8] for a detailed discussion. Here, *relative* weights are only implemented to enable comparisons.

`void SolveScaSta()`:
`void SolveScaSta(Int_t IScSta)`:
Combinations are performed while Sca-nning the estimator uncertainties using their Sta-tistical precision. For non of the active uncertainties the estimator correlations are allowed to be declared as `changed`, `scaled` or `reduced`, see `SetRho...Unc()` below for the definitions.

For each combination all active systematic uncertainties in all active estimates are changed according to the statistical precision of the uncertainties provided by the user when calling `FillSta()`. Since this is performed using the present list of active estimates the result may relate to several observables. If the result is wanted for a single observable, all estimates of the other observables should first be disabled using `SetInActiveEst()`

For all estimates i each uncertainty σ_{ik} is altered according to a Gaussian with mean zero and width equal to the statistical precision of the uncertainty source k, resulting in a value σ'_{ik}. In addition, depending on `IScSta`, also the correlation ρ_{ijk} of the pair (i, j) of estimates for the source k of uncertainty is changed. For the second implementation, the following options are implemented:
`IScSta = 0` means $\sigma'_{ik} = Max(0, \sigma'_{ik})$, ρ_{ijk} unchanged,
`IScSta = 1` means $\sigma'_{ik} = Abs(\sigma'_{ik})$, ρ_{ijk} unchanged,
`IScSta = 2` means $\sigma'_{ik} = Abs(\sigma'_{ik})$, ρ_{ijk} changed.
The default option `IScSta = 2` is used in the first implementation.

For `IScSta = 0` it is assumed that within its statistical precision, the uncertainty is (at most) allowed to vanish. For `IScSta = 1` it is assumed that within its statistical precision, the uncertainty is allowed to change *direction* as explained by the following example. Let the estimate i determine the top quark mass, and the uncertainty source k be the estimator uncertainty induced by the uncertainty in the experimental jet energy scale (JES). For the central result of this estimate it is assumed that, when applying the systematic uncertainty in the JES by increasing the nominal JES value by $+\sigma$(JES), the resulting top quark mass also increases, i.e. the JES uncertainty and the JES induced uncertainty in the top quark mass go into the same direction. However, within the limited statistical precision it is not excluded that, for a different but statistically compatible sample, the top quark mass decreases. Clearly, if this occurs for a dominant uncertainty, the statistical precision in the determination of that source should be improved. In contrast, for sub-dominant uncertainties this can be tolerated, but the implication on the resulting combined value should be assessed.

The option `IScSta = 2` is as `IScSta = 1` but in addition for cases with $\rho_{ijk} = \pm 1$ it is assumed that the correlation of the estimates is not assigned, but estimated as in Ref. [6]. In this case, ρ_{ijk} changes sign if one and only one of σ'_{ik}/σ_{ik} and σ'_{jk}/σ_{jk} is smaller than zero, i.e. one uncertainty has changed its direction, whereas the other has not.

The algorithm is implemented as follows. First a BLUE combination for the presently active estimates and uncertainties is performed yielding the original result and uncertainty. Then `NumSim = 500` additional combinations are performed and distributions of the $n = 1, \ldots,$ `NumSim` combined values and uncertainties are obtained for each observable. The results are stored in a TMatrixD of dimension `(GetActObs(),6)`, where the six values for each observable are:
0) the original combined value x,
1) the original uncertainty σ_x,
2) the mean combined value, $(\Sigma_n x_n)/$`NumSim`
3) the RMS of 2),

4) the mean uncertainty $(\Sigma_n \sigma_{x_n})/$`NumSim`, and
5) the RMS of 4).
The results can be printed using `PrintScaSta()` and retrieved using `GetStaRes()`.

The remaining two `Solve...()` functions implement alternative solving methods. *NOTE:* It is recommended to *NOT* use these functions when achieving scientific results because of the weaknesses of the concepts. See Ref. [5] for a detailed discussion of the consequences and a numerical example. Here, they are only implemented to enable comparisons.

`void SolvePosWei()`:
For each observable a combination is performed by including only estimates of this observable that have Pos-itive Wei-ghts and all other estimates of different observables. First a BLUE combination for the presently active estimates and uncertainties is performed. Then, all estimates that determine this observable, and have negative BLUE weights, are disabled and the next combination is performed. This is repeated until no estimates with negative weights remain.

`void SolveMaxVar(Int_t IFuRho)`:
This functions is only available for a single observable. Three methods are implemented to Max-imise the Var-iance of the combined result by changing, i.e. reducing the correlations of the systematic uncertainties in an artificial, but controlled way, see Ref. [8]. This is achieved by multiplying all covariance entries (i.e. the off diagonal elements of the contributions to the covariance matrix for the uncertainty source k) for $k > 0$ by factors f_{ijk}, thereby changing the initially assigned correlations. This procedure is not applied to the source $k = 0$, which is assumed to be the statistical uncertainty, which is either uncorrelated between estimates, or the correlations are exactly known, because they were determined by the experiments as e.g. in Ref. [9]. The following options are implemented:
`IFuRho =` `0` means $f_{ijk} = f$ for all i, j, k,
`IFuRho =` $\pm$ `1` means $f_{ijk} = f_k$ for all i, j,
`IFuRho =` `2` means $f_{ijk} = f_{ij}$ for all k.

Since for each source k of uncertainty and pair (i, j) of estimates the dependence of the relative improvement in the uncertainty follows Figure 2b, the factors f_{ijk} are obtained by a scan in the value of the respective factor using the range $1 \to 0$. The maximum is guaranteed to exist for $\rho_{ijk} = 1/z_{ijk} > 0$. Clearly, if the correlation initially assigned is such that it lies to the left of this point, the initial situation already corresponds to the maximum to be calculated, i.e. the real maximum is not attempted to be found in this procedure. See Ref. [5] for a detailed discussion of the consequences and a numerical example.

The algorithm is implemented as follows: For `IFuRho = 0`, the global factor f is found by a scan from $1 \to 0$. For `IFuRho = 1`, the f_k are obtained independently for all sources $k > 0$, i.e. when determining f_k the values for sources k' with $k' \neq k$ are set to unity. For `IFuRho = -1`, the f_k are obtained consecutively for all sources $k > 0$, i.e. when determining f_k the values for sources k' with $k' < k$ are set to their already found values. Finally, for `IFuRho = 2` the f_{ij} are found consecutively, while using the already determined values for $i' < i$ and $j' < j$. Given this procedure, the covariance matrix can be manipulated in such a way that the inversion gets unstable. The software is protected against this occurrence. Finally, the outcome can be digested by a call to `PrintMaxVar()`.

3.5 Setters

All setters are implemented in such a way that i and k always refer to their initial values for estimates and uncertainty sources that were given by the user during the filling step. This way the user does not need to keep track of the actual index an estimate or uncertainty has within the presently active list. The setters only work if the input is not fixed.

`void SetActiveEst(Int_t` i`)`:
Enable estimate i, i.e. it will be used in subsequent calls to `Solve()`.

`void SetActiveUnc(Int_t` k`)`:
Enable uncertainty k, i.e. it will be used in subsequent calls to `Solve()`.

`void SetInActiveEst(Int_t` i`)`:
Disable estimate i, i.e. it will not be used in subsequent calls to `Solve()`.

`void SetInActiveUnc(Int_t` k`)`:
Disable uncertainty k, i.e. it will not be used in subsequent calls to `Solve()`.

`void SetRhoValUnc(Double_t RhoVal)`:
`void SetRhoValUnc(Int_t` k`, Double_t RhoVal)`:
`void SetRhoValUnc(Int_t` k`, Int_t` ℓ`, Double_t RhoVal)`:
The first implementation of this function will set the correlations of all active uncertainty sources and all groups ℓ to `RhoVal`. This value should be within the range $-1 <$ `RhoVal` < 1. The second will do the same, but only for the source k. The third one only applies to the group ℓ of source k. See the constructor for the definition of the groups ℓ.

`void SetNotRhoValUnc()`:
`void SetNotRhoValUnc(Int_t` k`)`:
The first implementation of this function will revert to the originally provided correlations of all active uncertainty sources. The second will do the same, but only for the source k.

`void SetRhoFacUnc(Double_t RhoFac)`:
`void SetRhoFacUnc(Int_t` k`, Double_t RhoFac)`:
`void SetRhoFacUnc(Int_t` k`, Int_t` ℓ`, Double_t RhoFac)`:
The first implementation of this function will scale the originally provided correlations of all active uncertainty sources and all groups ℓ by a factor `RhoFac`. This factor should be within the range $-1 <$ `RhoFac` < 1. The second will do the same, but only for the source k. The third one only applies to the group ℓ of source k. See the constructor for the definition of the groups ℓ. Clearly, sources for which the estimators are uncorrelated are not affected by this.

`void SetNotRhoFacUnc():`
`void SetNotRhoFacUnc(Int_t` k`):`
The first implementation of this function will revert to the originally provided correlations of all active uncertainty sources. The second will do the same, but only for the source k.

The following functions implement the so called *reduced correlations*[1]. *NOTE:* It is recommended to *NOT* use these functions when achieving scientific results because of the weakness of the concept. See Ref. [5] for a detailed discussion of the consequences and a numerical example. Here, they are only implemented to enable comparisons.

`void SetRhoRedUnc():`
`void SetRhoRedUnc(Int_t` k`):`
For all active uncertainty sources and all **fully correlated** pairs of estimates, the first implementation of this function will replace the correlation by the reduced correlation. The second will do the same, but only for the source k.

`void SetNotRhoRedUnc():`
`void SetNotRhoRedUnc(Int_t` k`):`
The first implementation of this function will revert to the originally provided correlations of all active uncertainty sources. The second will do the same, but only for the source k.

By construction, `changed-`, `scaled-` and `reduced` correlations are mutually exclusive. Consequently, for each source of uncertainty the use of only one of the options is supported by the software.

The following functions allow to steer which uncertainties are taken as relative and which as absolute in subsequent calls to `SolveRelUnc(...)`.

`void SetRelUnc():`
`void SetRelUnc(Int_t` k`):`
The first implementation of this function will declare all active uncertainty sources as relative uncertainties. The second will do the same, but only for the source k. In this implementation the default behaviour of the detailed implementation discussed next is used for all estimates and the respective uncertainty source.

`void SetRelUnc(Int_t` i`, Int_t` k`, Double_t* ActCof):`
For each estimate i and each uncertainty source k the dependence of the variance on the combined value x is defined by using the coefficients from the array `ActCof` $= \{a_0, a_1, a_2\}$ in the second

[1]For each pair (i, j) of estimates and a given source of uncertainty k, reduced correlations split the uncertainty in two *independent* components. For the first, both estimators have an equal uncertainty assumed to be the smaller of the individual uncertainties, e.g. $\sigma_{1k} < \sigma_{2k}$, and are fully correlated. The second uncertainty is postulated to only apply to the estimator x_2, and the corresponding uncertainty is taken to be $\sqrt{\sigma^2_{2k} - \sigma^2_{1k}}$. This ad-hoc procedure replaces the covariance $\rho_{12k}\sigma_{1k}\sigma_{2k}$ by the square of the smaller of the individual uncertainties σ^2_{1k} for this source, which is equivalent to assuming the correlation to amount to the ratio of the smaller to the larger uncertainty, $\rho_{12k} = \sigma_{1k}/\sigma_{2k} = 1/z_k$.

order polynomial: $\sigma^2_{ik} = a_0 + a_1\,|x| + a_2\,x^2$.

In the default implementation it is assumed that the statistical uncertainty is proportional to $\sqrt{N}$ and the estimate to be proportional to N, where N is the number of events. Finally, the systematic uncertainties are assumed to be linear in $|x|$. Consequently, in this case only one coefficient each is different from zero. For the statistical uncertainty ($k = 0$) this is $a_1 = \sigma^2_{i0}/|x_\mathrm{i}|$, and for all systematic uncertainties $k > 0$ it is $a_2 = \sigma^2_{ik}/x^2_\mathrm{i}$. If this behaviour is valid for the combination under investigation, a single call to `void SetRelUnc()` should be used, otherwise individual user defined functions have to be provided. If this is needed, for any uncertainty source k the functions for all estimates i have to be given. The user should ensure that the coefficients are such that the functional form cannot lead to negative uncertainties, otherwise the combination cannot be performed and the input will not be fixed.

`void SetNotRelUnc()`:
`void SetNotRelUnc(Int_t` k`)`:
The first implementation of this function will declare all active uncertainty sources as absolute uncertainties and revert to the initially provided values. The second will do the same, but only for the source k.

3.6 Getters

3.6.1 Getters for estimates, uncertainties and observables

The following functions give access to the initial numbers of estimates, uncertainties and observables. This information is available after the constructor was called. They allow for tools to be written by the user, which only need the pointer to the BLUE object as input.

`Int_t GetNumEst() const`:
Returns the initial number of estimates.

`Int_t GetNumUnc() const`:
Returns the initial number of uncertainties.

`Int_t GetNumObs() const`:
Returns the initial number of observables.

3.6.2 Getters for active estimates, uncertainties and observables

The following functions give access to the actual numbers of active estimates, uncertainties and observables. Also for the observables the index n refers to the original index. This information is only available after a call to `FixInp()`, otherwise, if not stated differently, the return value is zero.

`Int_t GetActEst() const`:
Returns the number of active estimates.

`Int_t GetActEst(Int_t` n`) const`:
Returns the number of active estimates for the active observable n.

`Int_t GetActUnc() const`:
Returns the number of active uncertainties.

`Int_t GetActObs() const`:
Returns the number of active observables. Although the interface does not allow to disable observables, still this number will differ from the value of `NumObs` originally supplied, as soon as all estimates determining one of the observables were deactivated by calling `SetInActiveEst()`.

The following functions give access to the names of the active estimates, uncertainties and observables. This information is only available after a call to `FixInp()`, otherwise, as well as for inactive estimates, the return value is `NULL`.

`TString GetNamEst(Int_t` i`) const`:
Returns the name of the active estimate i.

`TString GetNamUnc(Int_t` k`) const`:
Returns the name of the active uncertainty k.

`TString GetNamObs(Int_t` n`) const`:
Returns the name of the active observable n.

The following functions give access to the actual lists of estimates, uncertainties and observables. Again, this information is only available after a call to `FixInp()`. In this case the return value is 1 otherwise it is 0. These functions also return a pointer to the first element of an array of `Int_t` values. The structures are filled always starting from element 0. The dimensions are dynamical, i.e. they depend on the number of active estimates, uncertainties and observables that may well differ from the dimensions originally supplied to the constructor of the class. As a consequence, if the structures are defined by the user and filled using the original dimensions, the last part of the structures will contain senseless values, whenever estimates or uncertainty sources are disabled and the functions are called a second time.

`Int_t GetIndEst(Int_t* IndEst) const`:
Returns the list of active estimates. The dimension is: `IndEst(GetActEst())`.

`Int_t GetIndUnc(Int_t* IndUnc) const`:
Returns the list of active uncertainties. The dimension is: `IndUnc(GetActUnc())`.

`Int_t GetIndObs(Int_t* IndObs) const`:
Returns the list of active observables. The dimension is: `IndObs(GetActObs())`.

`Int_t GetPreEst(Int_t` n`) const`:
Returns the index i of the most Pre-cise Est-imate for the active observable n. Because zero is a valid number for an estimate, in case of failure, a value of minus one is returned.

`Double_t GetCompatEst(Int_t` i`, Int_t` j`) const`:
Returns the compatibility of the pair i, j of active estimates. For the definition of the χ^2 used see the explanation of `PrintCompatEst()`. Because zero is a valid result, in case of failure, a value of minus one is returned.

The following functions give access to various quantities for the active estimates and uncertainties. See above for their availability and return values. These functions come in pairs and return a pointer to either a `TMatrixD` or the first element of an array of `Double_t` values. The structures are filled always starting from element $(0, 0)$ or 0. The dimensions of the matrices are given below, the dimension of the arrays should be the product of the number of columns and rows of the matrices. The user has to take care of the proper dimension of the structure in the calling function. Also here the dimensions are dynamical (see above for the consequences).

`Int_t GetEst(TMatrixD* UseEst) const`:
`Int_t GetEst(Double_t* RetEst) const`:
Returns the matrix of the active estimates in the form they were supplied in the user call to `FillEst()`. The dimension is: `UseEst(GetActEst(),GetActUnc()+1)`.

`Int_t GetEstVal(TMatrixD* UseEstVal) const`:
`Int_t GetEstVal(Double_t* RetEstVal) const`:
Returns the values of the active estimates.
The dimension is: `UseEstVal(GetActEst(),1)`.

`Int_t GetEstUnc(TMatrixD* UseEstUnc) const`:
`Int_t GetEstUnc(Double_t* RetEstUnc) const`:
Returns the values of the total uncertainties in the active estimates.
The dimension is: `UseEstUnc(GetActEst(),1)`.

`Int_t GetCov(TMatrixD* UseCov) const`:
`Int_t GetCov(Double_t* RetCov) const`:
Returns the covariance matrix of the estimates.
The dimension is: `UseCov(GetActEst(),GetActEst())`.

`Int_t GetCovInvert(TMatrixD* UseCovI) const`:
`Int_t GetCovInvert(Double_t* RetCovI) const`:
Returns the inverse of the covariance matrix of the estimates.
The dimension is: `UseCovI(GetActEst(),GetActEst())`.

`Int_t GetRho(TMatrixD* UseRho) const`:
`Int_t GetRho(Double_t* RetRho) const`:
Returns the matrix of the correlations of the estimates.

The dimension is: `UseRho(GetActEst(),GetActEst())`.

`Int_t GetCor(Int_t k, TMatrixD* UseCor) const`:
`Int_t GetCor(Int_t k, Double_t* RetCor) const`:
Returns the matrix of the correlations of the active estimates for the active uncertainty source k. The dimension is: `UseCor(GetActEst(),GetActEst())`.

`Int_t GetSta(TMatrixD* UseSta) const`:
`Int_t GetSta(Double_t* RetSta) const`:
Returns the statistical precision in all active sources of systematic uncertainty for all active estimates. The dimension is: `UseSta(GetActEst(),GetActUnc())`.

`Int_t GetCompatEst(TMatrixD* UseCompatEst) const`:
`Int_t GetCompatEst(Double_t* RetCompatEst) const`:
Returns the matrix of the compatibilities of the pairs of active estimates. For the definition of the χ^2 used see the explanation of `PrintCompatEst()`.
The dimension is: `UseCompatEst(GetActEst(),GetActEst())`.

`Int_t GetParams(Int_t ifl, TMatrixD* UseParams) const`:
`Int_t GetParams(Int_t ifl, Double_t* RetParams) const`:
Returns the matrices of parameters for hypothetical pairwise combinations. See `PrintParams()` for the meaning of `ifl`. The dimension is: `UseParams(GetActEst(),GetActEst())`.

3.6.3 Getters for the consistency of the combination

The following functions give access to information that is only available after a call to `Solve...()`, otherwise the return value is zero.

`Double_t GetChiq() const`:
Returns the χ^2 value of the result, i.e. the quantity minimised in the combination.

`Int_t GetNdof() const`:
Returns the number of degrees of freedom N_{dof}, i.e. the difference in the number of active estimates and active observables.

`Double_t GetProb() const`:
Returns the χ^2 probability $P(\chi^2, N_{\mathrm{dof}})$ of the result. The χ^2 probability is the integral of the χ^2 probability density function from the observed χ^2 value up to infinity. It constitutes the probability for an even larger χ^2 to occur for any other combination [10].

`Double_t GetPull(Int_t` i`) const`:
Returns the pull of the estimate i. The pull is defined as the difference in the estimate and the observable, divided by the square root of the difference in the variances of the two.

`Double_t GetCompatObs(Int_t` n`, Int_t` m`) const`:
Returns the compatibility of the pair n, m of active observables. For the definition of the χ^2 used see the explanation of `PrintCompatEst()`. Because zero is a valid result, in case of failure, a value of minus one is returned.

3.6.4 Getters for the results of the combination

The following functions give access to various quantities for the results for the active observables that are obtained from the combination of the active estimates given their active uncertainties. Again, this information is only available after a call to `Solve()`. Also here, this is indicated by the return value of the integer function, which is 1 if successful, i.e. `Solve()` was called, and 0 otherwise. These functions also come in pairs.

`Int_t GetCovRes(TMatrixD* UseCovRes) const`:
`Int_t GetCovRes(Double_t* RetCovRes) const`:
Returns the covariance matrix of the observables.
The dimension is: `UseCovRes(GetActObs(),GetActObs())`.

`Int_t GetRhoRes(TMatrixD* UseRhoRes) const`:
`Int_t GetRhoRes(Double_t* RetRhoRes) const`:
Returns the matrix of the correlations of the observables.
The dimension is: `UseRhoRes(GetActObs(),GetActObs())`.

`Int_t GetWeight(TMatrixD* UseWeight) const`:
`Int_t GetWeight(Double_t* RetWeight) const`:
Returns the matrix of the BLUE weights of the estimates of the various observables. The dimension is: `UseWeight(GetActEst(),GetActObs())`.

`Int_t GetResult(TMatrixD* UseResult) const`:
`Int_t GetResult(Double_t* RetResult) const`:
Returns the matrix of the results for the observables in the form expected for the filling of the estimates in `FillEst()` described above. Each observable is stored in one row, where the first element is the value, followed by the individual uncertainties. The dimension is: `UseResult(GetActObs(),GetActUnc()+1)`.

`Int_t GetUncert(TMatrixD* UseUncert) const`:
`Int_t GetUncert(Double_t* RetUncert) const`:
Returns the matrix of the total uncertainties in the observables.
The dimension is: `UseUncert(GetActObs(),1)`.

`Int_t GetCompatObs(TMatrixD* UseCompatObs) const`:
`Int_t GetCompatObs(Double_t* RetCompatObs) const`:
Returns the matrix of the compatibilities of the pairs of active observables. For the definition of the χ^2 used see the explanation of `PrintCompatEst()`.
The dimension is: `UseCompatEst(GetActObs(),GetActObs())`.

`Int_t GetInspectLike(TMatrixD* UseInsLik) const`:
`Int_t GetInspectLike(Double_t* RetInsLik) const`:
Returns the matrix containing the results of `InspectLike()`. See the description of `InspectLike()` for the matrix content. The dimension is: `UseUncert(GetActObs(),7)`.

3.6.5 Getters for specific solving methods

The following functions give access to various quantities for results of specific `Solve...()` methods. This information is only available after a call to the respective solver. Also here, this is indicated by the return value of the integer function. In case of failure, if not stated differently, the return value is zero.

`Int_t GetAccImpLasEst(Int_t` n`) const`:
Returns the index i of the last estimate Acc-ording to Imp-ortance to be used for the active observable n, based on the result of `SolveAccImp(..., Dx)`. Because zero is a valid number for an estimate, in case of failure, a value of minus one is returned.

`Int_t GetAccImpIndEst(Int_t` n`, Int_t* IndEst) const`:
Returns the list of estimates sorted Acc-ording to Imp-ortance for the active observable n, based on the result of `SolveAccImp()`. The dimension is: `IndEst[GetActEst(n)]`.

`Int_t GetNumScaFac() const`:
Returns the number of groups of correlations ℓ defined in the constructor.

`Int_t GetNumScaRho() const`:
Returns the number of steps in the correlations used in `SolveScaRho`, which is ten.

`Int_t GetScaVal(Int_t` n`, TMatrixD* UseScaVal) const`:
`Int_t GetScaVal(Int_t` n`, Double_t* RetScaVal) const`:
Returns the result of `SolveScaRho()` for the differences in the values of the observable n. Starting from $\ell = 0$, for each group, for all sources k the differences are reported in consecutive rows. The dimension is: `UseScaVal(GetActUnc()*GetNumScaFac(),GetNumScaRho())`.

`Int_t GetScaUnc(Int_t` n`, TMatrixD* UseScaUnc) const`:
`Int_t GetScaUnc(Int_t` n`, Double_t* RetScaUnc) const`:
Returns the result of `SolveScaRho()` for the differences in the uncertainties in the observable n in the same order as for `GetScaVal()`.
The dimension is: `UseScaUnc(GetActUnc()*GetNumScaFac(),GetNumScaRho())`.

`Int_t GetStaRes(TMatrixD* UseStaRes) const`:
`Int_t GetStaRes(Double_t* RetStaRes) const`:
Returns the result of `SolveScaSta()`. The meaning of the entries of `UseStaRes` are explained in the description of the solver. The dimension is: `UseStaRes(GetActUnc(),6)`.

3.7 Print-out

The software provides some print-out during the various steps. Naturally, printing more information helps developing the user functions, but afterwards it only distracts from the important information. Consequently, the level of details reported to the user can be steered.

`void SetPrintLevel(Int_t` p`)`:
Set the level of details for the print-out $0 \leq p \leq 2$, with increasing details for increasing values of p.

```
void SetFormat(TString ForUni) const:
void SetFormat(TString ForVal, TString ForUnc, TString ForWei,
               TString ForRho, TString ForPul, TString ForUni) const:
void SetFormat(TString ForVal, TString ForUnc, TString ForWei,
               TString ForRho, TString ForPul, TString ForChi, TString ForUni) const:
```

A number of printing functions use formats set to %5.2f for Val-ues and Unc-ertainties (and their optional statistical precision), and %4.2f for Wei-ghts, correlations (Rho), Pul-ls and Chi-square values. For better readability of some `Print...()` functions formats of the type %+n.mf are advantageous. The second implementation has only been kept for backward compatibilities. It may disappear in future releases. In addition the estimates frequently carry a Uni-t. The default value for this is simply set to `ForUni = "unit"`. If these are not suited, either only the unit, or both formats and unit can be defined by the user. The choice of `ForUni = "None"` results in no units to be printed in figures and tables.

`void SetNoRootSetup()`:
For producing figures some global ROOT variables are set by the software. However, this will overwrite any user style that is present when instantiating the object. This can be avoided by a call to this function before any call to `FixInp()`.

```
void SetLogo(TString LogNam, TString LogVer, Int_t LogCol):
void SetLogo(TString LogNam, TString LogVer, Int_t LogCol, Double_t* LogXva,
             Double_t* LogYva):
```

A logo, composed of two strings `LogNam, LogVer`, e.g. a name and a release will be added to the figures either at the position `LogXva, LogYva`, or at a dynamically chosen position, using the color `LogCol`. The information should be provided by a call to this function before any call to `FixInp()`.

`void SetQuiet()`:
On top of the general steering, there exist some print-out in `FixInp()` and `Solve()` that cannot be switched off by `SetPrintLevel()`. A call to this function will also switch off those, which is useful for iterative use of `Solve()`.

`void SetNotQuiet()`:
Will revert to the original print-out in `FixInp()` and `Solve()`.

On top of this, there exist five groups of `Print` functions.

- A group that simply prints a matrix or an array of Double_t values in a given format.
- A group that returns information related to the presently active estimates and uncertainties. Given the flexibility of the `Set` functions described above, they only reflect the correct status after a call to `FixInp()`. Consequently, they are disabled until this function was called.
- A group that returns information related to the result for the observables. They only reflect the correct status after a call to `Solve()`. Consequently, they are disabled until this function was called.
- A group that consists of just one function that shows the present status of the input and the results. Depending on the print level it calls a number of functions from the above groups.
- A group that returns the finding of specific `Solve...()` functions described above. Again they are only available after the respective solver was called.

These functions are described in the following Sections.

3.7.1 Print functions for matrices and arrays

```
void PrintMatrix(TMatrixD* TryMat) const:
void PrintMatrix(TMatrixD* TryMat, TString ForVal) const:
void PrintMatrix(TMatrixD* TryMat, Int_t NumRow, Int_t NumCol) const:
void PrintMatrix(TMatrixD* TryMat, Int_t NumRow, Int_t NumCol, TString ForVal) const:
void PrintMatrix(TMatrixD* TryMat, Int_t MinRow, Int_t MaxRow, Int_t MinCol,
                 Int_t MaxCol) const:
void PrintMatrix(TMatrixD* TryMat, Int_t MinRow, Int_t MaxRow, Int_t MinCol,
                 Int_t MaxCol, TString ForVal) const:
```

The first pair of functions prints the matrix `TryMat`, where the format is `ForVal` if provided, or %5.2f otherwise, and where the numbers of rows and columns are derived from the matrix. For the second pair, the numbers of rows `NumRow` and columns `NumCol` printed can be restricted to smaller values, but always starting from `TryMat(0,0)`. Finally, for the last pair, the printout is restricted to `MinRow` $\leq$ `Row` $\leq$ `MaxRow` and `MinCol` $\leq$ `Column` $\leq$ `MaxCol`.

```
void PrintDouble(Double_t* TryDou, Int_t NumRow, Int_t NumCol) const:
void PrintDouble(Double_t* TryDou, Int_t NumRow, Int_t NumCol, TString ForVal) const:
```

Same as `PrintMatrix()`, but for an array of Double_t values. Given the array is one dimensional, the numbers of rows and columns have to be specified.

Internally, the array is stored in a matrix and `PrintMatrix()` is called. If the additional functionality of `PrintMatrix()` is wanted, the user should store the array in a matrix by `TMatrixD* TryMat = new TMatrixD(NumRow, NumCol, &TryDou[0])` and use `PrintMatrix()`.

3.7.2 Print functions for active estimates

Again, also for the print functions, the indices i and k always refer to their initial values for estimates and uncertainty sources that were given by the user during the filling step.

`void PrintListEst() const`:
Prints the list of the active estimates.

`void PrintListUnc() const`:
Prints the list of the active uncertainties.

`void PrintNamEst() const`:
Prints the names of the active estimates.

`void PrintNamUnc() const`:
Prints the names of the active uncertainties.

`void PrintEst(Int_t` i`) const`:
Prints the information for the active estimate i.

`void PrintEst() const`: Prints the information for all active estimates.

`void PrintCofRelUnc() const`:
`void PrintCofRelUnc(Int_t` k`) const`:
The first implementation of this function will print for all active estimates the coefficients for all active and relative uncertainties. The second will do the same, but only for the uncertainty source k.

`void PrintCor(Int_t` k`) const`:
Prints the matrix of the estimator correlations for the active uncertainty source k.

`void PrintCor() const`:
Prints matrix of the estimator correlations for all active uncertainty sources.

`void PrintCov(Int_t` k`) const`:
Prints the contribution of the uncertainty source k to the covariance matrix of the active estimates.

`void PrintCov() const`:
Prints the covariance matrix of the active estimates.

`void PrintCovInvert() const`:
Prints the inverted covariance matrix of the active estimates.

`void PrintRho() const`:
Prints matrix of the estimator correlations for the active estimates.

`void PrintCompatEst() const`:
`void PrintCompatEst(TString FilNam) const`:
Prints the pair wise compatibility of the estimates of the same observable given their correlation. The compatibility is based on a χ^2 and the corresponding probability using $P(\chi^2, N_{\mathrm{dof}} = 1)$. The χ^2 is defined as:

$$\chi^2 = \frac{(x_1 - x_2)^2}{\sigma_1^2 + \sigma_2^2 - 2\,\rho_{12}\,\sigma_1\,\sigma_2}.$$

For a detailed discussion see Ref. [5]. If the second implementation is used, the χ^2 and $P(\chi^2, 1)$ distributions for all observables are stored in two files called `FilNam_ComEst_ChiQua.pdf` and `FilNam_ComEst_ChiPro.pdf` An example is shown in Figure 4.

`void PrintParams(Int_t ifl) const`:
Prints the matrices of parameters for hypothetical pairwise combinations of the estimates i and j provided they determine the same observable. If not, for this pair zero is returned instead. Given the symmetry, only the lower half of the matrix is filled. The parameter printed depends on the value of `ifl`. For `ifl = 0` the ratio of the uncertainties is returned, i.e. σ_i/σ_j with $j > i$. This ratio corresponds to z if $\sigma_i > \sigma_j$ and $1/z$ otherwise. For $1 \leq$ `ifl` ≤ 6 the result of Eqs. 2–7 is returned.

`void PrintPull(Int_t` i`) const`:
Prints the pull of the active estimate i.

`void PrintPull() const`:
Prints the pull of all active estimates.

3.7.3 Print functions for active observables

`void PrintListObs() const`:
Prints the list of active observables.

`void PrintNamObs() const`:
Prints the names of the active observables.

`void PrintCovRes() const`:
Prints the covariance matrix of the results for all observables.

`void PrintRhoRes() const`:
Prints the correlation matrix of the results for all observables.

`void PrintWeight() const`:
Prints the weight matrix of the results for all estimates (rows) and observables (columns), see `GetWeight()`.

`void PrintResult() const`:
Prints the result for each observable. First the linear combination of the individual estimates is given. Then the combined values for the observables are listed together with the full breakdown of their uncertainties. The uncertainties are calculated from Eq. 18 of Ref. [2]. Because the minimisation is performed based on the total uncertainties and estimator correlations, it is not guaranteed that all variances σ^2_{nk} of individual uncertainties k of the observables n from Eq. 18 are positive. In rare cases $\sigma^2_{nk} < 0$ occurs. This is indicated by quoting $\sigma_{nk} = -\sqrt{-\sigma^2_{nk}}$ as uncertainty. It is worth noting that this only indicates that for this source the interpretation as an uncertainty is not meaningful. However, the combined result is not affected and thus trustworthy.

Finally, the separation into the statistical uncertainty ($k = 0$) and the total systematic uncertainty (the square root of the difference in quadrature of the total uncertainty and the statistical uncertainty) is given.

`void PrintCompatObs() const`:
Prints the pair wise compatibility of the observables given their correlation. For the definition of the χ^2 used see the explanation of `PrintCompatEst()`. Obviously, here this compatibility only makes sense if the observables should coincide, i.e. they relate to the same physics parameter. If that is not the case this information should be ignored.

`void PrintChiPro() const`:
Prints the χ^2 together with the number of degrees of freedom N_{dof}, and the χ^2 probability $P(\chi^2, N_{\mathrm{dof}})$ of the result, see the corresponding `Getters` for the definitions.

`void PrintInspectLike() const`:
Prints the results from `InspectLike(`n`)`. For each observable n for which `InspectLike(`n`)` was called, the result from the likelihood and the BLUE method are listed.

3.7.4 Print functions for the overall status

`void PrintStatus() const`:
Prints the status of input and output depending on the state like: `fixed` or `solved` and the `print level`.

3.7.5 Print functions for specific solving methods

`void PrintAccImp() const`:
Prints the findings of `SolveAccImp(..., Dx)`. The order of importance is given. For each observable, the parameters for the hypothetical pairwise combinations of each of its estimate with the most precise estimate is given. The change is reported for the combined value and its uncertainty while including the estimates one by one according to importance in the combination. Finally, the list of estimates to be used in the combination is given that corresponds to

the relative improvement `Dx` requested.

`void PrintScaRho() const`:
`void PrintScaRho(TString FilNam) const`:
Prints the differences in the values and uncertainties in all observables obtained in the correlation scan performed by `SolveScaRho()`. The matrix with the remaining correlation groups ℓ is given. The ranges in r used are listed per uncertainty source k, and group of correlation ℓ. The number of inversion failures is reported if present. For each observable the differences in the values and uncertainties are reported per source k and for all ten values of r. Finally, the total differences are reported with the following meaning. For the independent scan, i.e. `RhoFla = 0` the total is the quadratic sum of all sources ignoring inversion failures, i.e. entries reported as -1.00. In contrast, for the simultaneous scan, i.e. `RhoFla = 1`, the total coincides with the last line of the previously accumulated result.

When the second implementation is used the result of the scan is displayed in a pair of figures per observable and group of estimates. These figures contain the observed shifts in the values and uncertainties respectively for three steps of the scan, namely step four, seven and ten. The names of the files are `FilNam_ScaRho_XxxYyy_Zzz_Obs_N.pdf`. Here `Xxx` is `Ind` for independent variations per source k of uncertainty, i.e. `RhoFla = 0`, and `Sim` for simultaneous variations, i.e. `RhoFla = 1`. In addition, `Yyy` is `Mor` if there are more than one group ℓ of estimates to be scanned, and `One` otherwise. Finally, `Zzz` is either `Val` for the values or `Unc` for the uncertainties, and `N` is the value of n stored in the format %i. An example is shown in Figure 5.

`void PrintInfWei() const`:
Prints the information weights defined above in the description of `SolveInfWei()` for the active estimates.

`void PrintScaSta() const`:
`void PrintScaSta(TString FilNam) const`:
`void PrintScaSta(TString FilNam, Double_t MinVal, Double_t MaxVal,`
`                 Double_t MinUnc, Double_t MaxUnc) const`:
Prints the findings of `SolveScaSta()`. If the second implementation is used, the distributions of the combined values `Val` and uncertainties `Unc` for all observables `Obs` are stored in pairs of files called `FilNam_ScaSta_M_Val_Obs_N.pdf` and `FilNam_ScaSta_M_Unc_Obs_N.pdf`, where `M` is the value of `IScSta` stored in the format %i, and `N` the value of n stored in the format %i. For an example see Figure 6.

`void PrintMaxVar() const`:
Prints the findings of `SolveMaxVar()`. The variance of the combined result and the correlation matrix of the estimates are given before and after the maximisation of the variance. In addition listed are the number of times an unstable matrix inversion was detected. Finally, the calculated factors are given, depending on the value of `IFuRho` used in the call to `SolveMaxVar(IFuRho)`.

3.8 Utilities

For the special situation of two estimates of a single observable discussed above, the data can be inspected more closely. Two sets of functions are implemented. The first set is independent of the data structure. The second set (at present containing only a single function) works on a pair of active estimates. Both sets are discussed in turn.

For the situation of relative uncertainties, `SolveRelUnc()` is only an approximate solution. A utility is provided to compare this to the result obtained from a simplified maximum likelihood approach. In addition, a utility is provided to inspect the situation in the case of instable matrix inversions. Finally, for publishing the results a number of utilities to create LaTeX and PDF output are provided. These utilities are discussed in turn.

3.8.1 Data structure independent utilities for a pair of estimates

The first two functions can be used for arbitrary values of ρ and z, to either evaluate Eqs. 2–7, or to produce figures analogous to Figures 2a–3d.

`Double_t GetPara(Int_t ifl, Double_t rho, Double_t zva) const`:
Returns for given values of $rho = \rho$ and $zva = z$ the values of Eq. `ifl+1`.

`Double_t FunPara(Double_t* x, Double_t* par) const`:

This function implements the possibility to use `GetPara()` as a TF1 function. The meaning of the parameters is as follows: for all cases `par[1] = ifl`. For the situation that z is a parameter, and ρ is the function variable, as e.g. in Eq. 2, `par[0]` $= z$ and `x[0]` $= \rho$. For the situation that ρ is a parameter, and z is the function variable, as e.g. in Eq. 7, the situation is reversed, i.e. `par[0]` $= \rho$ and `x[0]` $= z$.

For the user to implement this as a TF1 function the following notation should be used:
TF1* Func = new TF1(FuncName,this,&Blue::FunPara,xlow,xhig,2,"Blue","FunPara");
Only when this syntax is followed the normal ROOT methods for TF1 functions can be used.

Finally, the last utility exploits the characteristics of an arbitrary pair of estimates.

```
void DrawSens(Double_t xv1, Double_t xv2, Double_t sv1, Double_t sv2,
              Double_t rho, TString FilNam) const:
void DrawSens(Double_t xv1, Double_t xv2, Double_t sv1, Double_t sv2,
              Double_t rho, TString FilNam, Int_t IndFig) const:
```

The required input is $xv1 = x_1$, $xv2 = x_2$, $sv1 = \sigma_1$, $sv2 = \sigma_2$, $rho = \rho$, see Section 1 for details. The same up and down variations of ρ and z as discussed below for `InspectPair()` are performed and visualised in a figure. For an example see Figure 7. This figure is finally stored in the file `FilNam_InsPai.pdf`. For the first implementation, i.e. for `IndFig` $= 0$ only a combined figure is shown, for `IndFig` $= 1$ also individual figures are drawn and stored in the files `FilNam_InsPai_X.pdf`, with `X = a, ..., h`.

3.8.2 Data structure dependent utility for a pair of estimates

`void InspectPair(Int_t` i`,Int_t` j`) const`:
`void InspectPair(Int_t` i`,Int_t` j`, TString FilNam) const`:
`void InspectPair(Int_t` i`,Int_t` j`, TString FilNam, Int_t IndFig) const`:
The pair of active estimates i, j is inspected more closely. When the second or third implementation is invoked also `DrawSens` (see above) is called for this pair. The name of the output file will be `FilNam_xi_xj_InsPai.pdf` where `i` and `j` are the value of the estimates i and j stored in the format %i.

First, the compatibility of the estimates is evaluated. If the estimates are not consistent, no combination should be performed, see Ref. [5] for a detailed discussion of this issue. Then the actual combination is performed and the values of Eqs. 2–7 are reported. Subsequently, the parameters ρ and z are varied by about $\pm 10\%$ in the following way. A variation of ± 0.1 is attempted in ρ. In addition, the variation is restricted to stay within $-0.99 < \rho < 0.99$ such that, depending on the initial value of ρ, the actual range may be smaller. Similarly, for z an upward variation to $z_{\rm up} = 1.1 \cdot z$ is performed. The downward variation to $z_{\rm dn} = 0.9 \cdot z$ is further restricted to not fall below the minimum of $z_{\rm dn} = 1.01$. This ensures that x_1 remains the more precise estimate. The combination is repeated for all possible pairs of values using the three cases each for $(z_{\rm dn}, z, z_{\rm up})$ and $(\rho_{\rm dn},\ \rho,\ \rho_{\rm up})$. All nine results and the observed range in x and σ_x are reported.

3.8.3 Utility to compare to the maximum likelihood approach

`void InspectLike(Int_t` n`) const`:
`void InspectLike(Int_t` n`, TString FilNam) const`:
This function is only available after the problem was solved by any of the `Solve...()` methods. It inspects the result of a likelihood fit for the observable n. For the second implementation, the result will also be stored in file called `FilNam_InsLik_Obs_N.pdf`, where `N` is the value of n stored in the format %i. The findings can be printed using `PrintInspectLike()`.

In principle, for a pair of estimates, the most likely true value $x_{\rm T}$ can be obtained from a maximum likelihood fit to Eq. 1, in which for each value of $x_{\rm T}$ the corresponding value for $\sigma_i(x_{\rm T})$ is used, i.e. in which also the non Gaussian nature is taken into account. This equation can be generalised to `NumEst` estimates and `NumObs` observables. There exist dedicated software package that implement the multi-dimensional maximum likelihood method. Here, for the purpose of investigating the quality of the approximation of `SolveRelUnc()`, a more simple one-dimensional approach per observable n is used. Using Eq. 1, for each estimate `i` determining this observable, the difference in the numerator takes the correct form $x_{\rm i} - x_{\rm T}$. In contrast for the remaining estimates, which determine observables $m \neq n$ the numerator is replaced by $x_{\rm i} - x$, i.e. the combined value from the BLUE method is used instead of the true value of that observable. This retains the correlations to the estimates that do not determine the observable under investigation, but reduces the likelihood to a one-dimensional function of $x_{\rm T}$. Clearly, when only combining estimates determining the same observable, this approximation is exact.

The method is implemented as follows. After performing the BLUE combination, for each observable, the corresponding likelihood is constructed and maximised. The results achieved this way are compared to the ones from the BLUE method, i.e. $x_{\rm T}$ is compared to x. When using any solver but `SolveRelUnc()`, the uncertainties are Gaussian, and the maximum likelihood and the BLUE results coincide. Otherwise they in principle differ, see Ref. [5] for a detailed discussion and a number of examples. Finally, the results are stored in a matrix that contains one row per observable. Within each row the results are listed in the following order: $x_{\mathrm{T}}, x_{\mathrm{T,low}}, x_{\mathrm{T,high}}, x, x_{\mathrm{low}}, x_{\mathrm{high}}$, `LikFla`, i.e. the result of the likelihood together with its uncertainties is listed first, followed by the corresponding numbers for the BLUE method. Finally, the value of `LikFla` has the following meaning:

`LikFla =` 1 one active observable, and at least one relative uncertainty
`LikFla =` 2 several active observables and at least one relative uncertainty
`LikFla =` 11 one active observable, and no relative uncertainties
`LikFla =` 12 several active observables and no relative uncertainties

It is worth noticing that the observed size of the differences for a specific combination is of limited importance. It explicitly only applies to the present set of estimates under study. It has no general meaning for the unknown underlying multi-dimensional pdf, but signals that for the particular case the choice of the method of combination matters, see Ref. [5] for a detailed discussion.

3.8.4 Utility to inspect instable matrix inversions

`Int_t InspectResult() const`:
For some input, or when manipulating individual elements of the covariance matrix unusual situations can occur. The return value of the function indicates which of the situations occurred. Starting from an initial return value of zero, the values given within brackets below are added to it. At present, four non exclusive situations are distinguished. Firstly, an individual uncertainty in an observable gets negative (+1) or its evaluation results in a `-nan` value (+10), secondly the same happens to the total uncertainty in an observable (+100 or +1000), thirdly the total uncertainty in an observable is larger than the one of its most precise estimate (+10000) and fourthly the matrix has negative eigenvalues (+100000). Finally, the negative value of the accumulated result is returned. For a user call to `Solve()`, in any of these cases a message is issued by the software. If this occurs, the situation can be inspected by setting the print level to greater than zero and calling `InspectResult()`, which will also report the occurrence of negative Eigenvalues of the covariance matrix if present.

3.8.5 Utilities for publishing

```
void LatexResult(TString FilNam) const:
void LatexResult(TString FilNam, TString ForVal, TString ForUnc, TString ForWei,
                 TString ForRho, TString ForPul) const:
void LatexResult(TString FilNam, TString ForVal, TString ForUnc, TString ForWei,
                 TString ForRho, TString ForPul, TString ForChi) const:
```
Creates a LaTeX file `FilNam.tex` with a number of tables. The tables provided are: a table containing the active estimates together with the observables, both with the individual uncertainties and for the estimates also the statistical precision in the systematic uncertainties, if present. In addition, tables are produced containing the correlations of the estimates for each source of uncertainty and for the total uncertainty; a table containing the compatibilities of all pairs of estimates determining the same observable; a table containing the blue weights and pulls. For

`NumObs>1`, additional tables are provided containing the compatibilities and correlations of all pairs of observables.

The first implementation uses default formats `ForXxx` where `Xxx` stands for the Val-ues, Unc-ertainties, Wei-ghts, Correlations (Rho), Pul-ls, and finally the Chi-square values, see `SetFormat()`. If these are not suitable for the case under study they can be individually provided by the user using the second implementation. After creation, this file can be processed from the shell using the local LaTeX implementation.

`void DisplayPair(Int_t` i`, Int_t` j`, TString FilNam) const`:
`void DisplayPair(Int_t` i`, Int_t` j`, TString FilNam Double_t MinVal, Double_t MaxVal,`
`Double_t MinUnc, Double_t MaxUnc) const`:
`void DisplayPair(Int_t` i`, Int_t` j`, TString FilNam, TString ForVal, TString ForUnc,`
`TString ForRho) const`:
`void DisplayPair(Int_t` i`, Int_t` j`, TString FilNam, Double_t MinVal, Double_t MaxVal,`
`Double_t MinUnc, Double_t MaxUnc, TString ForVal, TString ForUnc,`
`TString ForRho) const`:

Displays the result for an active Pair (i, j) of estimates. This function produces figures that are similar to the ones for β and σ_x/σ_1 produced by `DrawSense()`, but instead uses the absolute values x and σ_x. A function `FilNam_DisPai_i_j.cxx` is created, where `i` and `j` are the indices of the estimates i and j stored in the format %i. This function, after compiling (see Section 4) produces two files. The first, `FilNam_xi_xj_DisPai_Val.pdf`, contains the two estimates, their combination x and the possible results as a function of ρ. The second, `FilNam_xi_xj_DisPai_-Unc.pdf`, contains the two uncertainties, the uncertainty in the combination σ_x and the possible uncertainties as a function of ρ. An example is shown in Figure 9. The horizontal axes always range from -1 to 1. The variables `MinVal` and `MaxVal` allow to set the boundaries of the vertical axis in the figure showing x. Similarly, the variables `MinUnc` and `MaxUnc` allow to set the boundaries of the vertical axis in the figure showing σ_x. If they are not provided the boundaries are dynamically determined from the pair of estimates. For the definition of the formats `ForXxx` see the description of `SetFormat()`. If these are not suitable for the case under study they can be individually provided by the user using the third or fourth implementation.

`void CorrelPair(Int_t` i`, Int_t` j`, TString FilNam) const`:
`void CorrelPair(Int_t` i`, Int_t` j`, TString FilNam, TString ForUnc) const`:
`void CorrelPair(Int_t` i`, Int_t` j`, TString FilNam, Double_t XvaMin, Double_t XvaMax,`
`Double_t YvaMin, Double_t YvaMax) const`:
`void CorrelPair(Int_t` i`, Int_t` j`, TString FilNam, Double_t XvaMin, Double_t XvaMax,`
`Double_t YvaMin, Double_t YvaMax, TString ForUnc) const`:

Displays the Correl-ation for an active Pair (i, j) of estimates. This function produces a figures similar to Figure 3 of Ref. [11]. A function `FilNam_xi_xj_CorPai.cxx` is created, where `i` and `j` are the value of the estimates i and j stored in the format %i. This function, after compiling (see Section 4) produces the file `FilNam_xi_xj_CorPai.pdf` containing those uncertainties of the pair of estimates for which the estimator correlation is $\rho \pm 1$. The uncertainties are given together with their statistical precision, see `FillSta()`, if present. An example is shown in Figure 10.

`void DisplayResult(Int_t` n`, TString FilNam) const`:
`void DisplayResult(Int_t` n`, TString FilNam, TString ForVal, TString ForUnc) const`:

Displays the result for an active observable n. A function `FilNam_DisRes_Obs_N.cxx` is created, where `N` is the value of n stored in the format %i. This function, after compiling (see Section 4) produces a file `FilNam_DisRes_Obs_N.pdf` with a figure containing the active estimates that determine the observable n together with the result of the combination. For the definition of the formats `ForXxx` see the description of `SetFormat()`. An example is shown in Figure 11.

`void DisplayAccImp(Int_t` n`, TString FilNam) const`:
`void DisplayAccImp(Int_t` n`, TString FilNam, TString ForVal, TString ForUnc) const`:
Displays the result of `SolveAccImp(..., Dx)` for an active observable n. A function `FilNam_-AccImp_Obs_N.cxx` is created, where `N` is the value of n stored in the format %i. This function, after compiling (see Section 4) produces a file `FilNam_AccImp_Obs_N.pdf` with a figure containing the results of the successive combinations of `SolveAccImp(Dx)` for the observable n. In this figure, the combined result corresponding to the suggested list of estimates given the value of `Dx` is shown in red. For the definition of the formats `ForXxx` see the description of `SetFormat()`. An example is shown in Figure 12.

4 Examples

To demonstrate the usage of the software a number of example functions are provided. They reproduce the numerical values of all combinations performed in the respective publication (but for differences that are explained below). In some cases a few more combinations are performed based on the information contained in the original publications. In addition, the functions show examples of how to perform specific combinations and how to retrieve the results into local data structures. Using `grep Shows B_*cxx|grep ThisFunction` indicates examples that contain the usage of the function `ThisFunction`. Inspecting the respective example and copying code snippets hopefully helps in writing software using BLUE. The following examples in alphabetical order are provided:

`B_arXiv_1107_5255.cxx(Int_t Flag)`:
Function that reproduces the 2011 (v3) combination of the Tevatron results on the top quark mass [12].

`B_arXiv_1305_3929.cxx(Int_t Flag)`:
Function that reproduces the 2013 (v2) combination of the Tevatron results on the top quark mass [13].

`B_arXiv_1307_4003.cxx(Int_t Flag)`:
Function that reproduces the results in Ref. [8]. (A different minimum with respect to the one quoted in Table 6 is found for the maximisation of the variance for `IFuRho = 3`. See the printout of the example for further details.)

`B_arXiv_1403_4427.cxx(Int_t Flag)`:
Function that reproduces the 2014 combination of the Tevatron and LHC results on the top quark mass [14].

`B_arXiv_1407_2682.cxx(Int_t Flag)`:
Function that reproduces the 2014 combination of the Tevatron results on the top quark mass [15]. (There is a typo for the uncertainty in the top quark mass stemming from the lepton modelling quoted in Table 3. The value should read 0.01 rather than 0.07, i.e. the value found is 0.007. This has a negligible impact and was confirmed by the authors.)
`B_arXiv_1608_01881.cxx(Int_t Flag)`:
Function that reproduces the 2016 combination of the Tevatron results on the top quark mass [16].

`B_ATLAS_CONF_2012_095.cxx(Int_t Flag)`:
Function that reproduces the 2012 combination of the LHC results on the top quark mass [17].

`B_ATLAS_CONF_2012_134.cxx(Int_t Flag)`:
Function that reproduces the 2012 combination of the LHC results on the cross-section of top quark pair production [18].

`B_ATLAS_CONF_2013_033.cxx(Int_t Flag)`:
Function that reproduces the 2013 combination of the LHC results on the W-Boson polarisation in top quark pair events [19]. (Some discrepancies with respect to the published Tables 6 and 7 were found and are under investigation with the authors.)

`B_ATLAS_CONF_2013_098.cxx(Int_t Flag)`:
Function that reproduces the 2013 combination of the LHC results on the single top quark cross-section in the t-channel [20] using the BLUE method with relative uncertainties.

`B_ATLAS_CONF_2013_102.cxx(Int_t Flag)`:
Function that reproduces the 2013 combination of the LHC results on the top quark mass [21]. (A typo for the χ^2 value quoted in Table 4 was found and was acknowledged by the authors.)

`B_ATLAS_CONF_2014_012.cxx(Int_t Flag)`:
Function that reproduces the 2014 combination of the LHC results on the $t\bar{t}$ charge asymmetry [22]. (A typo was found for the correlation assumption for the W+jet modelling quoted in Table 1 that should read 100%. This was acknowledged by the authors. However, using the quoted 50% instead would have a very small numerical impact.)

`B_ATLAS_CONF_2014_052.cxx(Int_t Flag)`:
Function that reproduces the 2014 combination of the LHC results on cross-section measurements for associated production of a single top quark and a W-Boson [23].

`B_ATLAS_CONF_2014_054.cxx(Int_t Flag)`:
Function that reproduces the 2014 combination of the LHC results on the cross-section of top quark pair production in the eμ final state [24].

`B_CMS_PAS_2014_015.cxx(Int_t Flag)`:
Function that reproduces the 2014 combination of the CMS results on the top quark mass [25]. A number of differences were found and have been passed on to the authors.

`B_EPJC_72_2046.cxx(Int_t Flag)`:
Function that reproduces all results discussed in Ref. [9].
`B_EPJC_74_3004.cxx(Int_t Flag)`:
Function that reproduces the results of Table 2 of Ref. [5]. The results from Table 1 and 3 of Ref. [5] can be obtained using `B_Peelles.cxx(0-4)` and `B_arXiv_1305_3929.cxx(1)`. The results for the comparisons of absolute and relative uncertainties listed in the text of Section 5 of Ref. [5] are provided by running the corresponding examples.

`B_EPJC_74_3109.cxx(Int_t Flag)`:
Function that repeats the likelihood combination of the pole mass measurements from Ref. [26] using the BLUE method.

`B_EPJC_75_330.cxx(Int_t Flag)`:
Function that reproduces the results of Table 3 of Ref. [6]. This combination, based on the determined correlations of the estimates of the top quark mass for all sources of systematic uncertainty, is compared to the one using the traditional assignment of correlations.

`B_EPJC_79_290.cxx(Int_t Flag)`:
Function that reproduces the results of Ref. [27].

`B_JHEP_03_176.cxx(Int_t Flag)`:
Function that reproduces the results of Ref. [28].

`B_JHEP_10_072.cxx(Int_t Flag)`:
Function that reproduces the results of Ref. [29].

`B_NegVar.cxx(Int_t Flag)`:
Function that shows how negative variances for specific *sources of uncertainty* can occur. This is not a numerical problem of the combination but a feature of the input. See the output of the example for more details.

`B_NIMA_270_110.cxx()`:
Function that reproduces all results discussed in Ref. [1].

`B_NIMA_500_391.cxx(Int_t Flag)`:
Function that reproduces all results discussed in Ref. [2].

B_Peelles.cxx(): Function that reproduces Peelle's Puzzle, see Refs. [30, 31][2], and the additional scenarios discussed in Ref. [5].

B_PLB_761_350(Int_t Flag):
Function that reproduces the results of Table 2 of Ref. [11]. This combination, based on the determined correlations of the estimates of the top quark mass for all sources of systematic uncertainty, is compared to the one using the traditional assignment of correlations.

B_PLB_784_345(Int_t Flag):
Function that compares the combination of the Higgs boson mass from Ref. [32] to a BLUE combination with some assumptions on the correlations.

B_PRD41_982.cxx(Int_t Flag):
Function that reproduces the combination of Ref. [7] using the BLUE method with individual relative uncertainties.

B_PRD79_092005.cxx(Int_t Flag):
Function that reproduces the likelihood combination of Ref. [33] using the BLUE method to get an indication of the correlation of the two measurements.

B_PRD88_052018.cxx(Int_t Flag):
Function that reproduces the combination of the Tevatron measurements of the W-Boson mass of Ref. [34] using the BLUE method with and without reduced correlations.

B_PRD_93_072004.cxx(Int_t Flag):
Function that reproduces the published 2014 combination of the CMS results on the top quark mass [35]. A number of differences were found and have been passed on to the authors.

B_PRL_114_191803.cxx(Int_t Flag):
Function that compares the combination of the Higgs boson mass from Ref. [36] to a BLUE combination with some assumptions on the correlations.

For each example B_name.cxx a file B_name.inp is provided that enables the creation of an output file for that example by typing: `root -b < B_name.inp > B_name.out`. To further ease the usage, two shell scripts BlueOne and BlueAll are provided. A single example is run by typing `BlueOne B_name` at the shell prompt. To use all input files B_name.inp in the current directory simply type BlueAll at the shell prompt. To verify the absence of programming mistakes within the user software that can be detected by the compiler also CompOne and CompAll are provided. They should be used in an analogous way to BlueOne and BlueAll, but this time to compile B_name.cxx. In addition, LtexOne and LtexAll are provided. They should be used in an analogous way to BlueOne and BlueAll, but this time to run LaTeX on one or all B_name*.tex files. Finally, CleaOne and CleaAll are provided. They remove all output from the BLUE software for the file B_name.

[2]The puzzle was introduced in an internal memorandum [30]. The originally used numerical values can be found in Ref. [31].

5 Conversion of input files

To facilitate the conversion for users that have been working with the Fortran software [3], a utility is provided that takes a corresponding ASCII input file and converts it to a function that is similar to the examples listed above.

`void ForttoBlue(TString FilNam, TString ForVal, TString ForRho) const`:
This function uses the input file `FilNam.in` and creates a file `B_FilNam.cxx` together with a corresponding steering file `B_FilNam.inp`. Afterwards `B_FilNam.cxx` can be expanded by the user and finally, it should be used the same way as the examples described in Section 4.

Running the Fortran software on `FilNam.in` should give the same result than what is obtained using `B_FilNam.cxx`. The format statement `ForVal` applies to the write statements for the estimates and uncertainties, and `ForRho` to the entries in the matrices of the estimator correlations. See `SetFormat()` for a more detailed description of the meaning. Since this utility performs formatted reading from a file, strict requirements on the content of `FilNam.in` are imposed, e.g. blanks in names are not supported. The full list of requirements is listed when running `ForttoBlue()`. The function `ForttoBlue()` reports the findings during execution, such that in the case of failures the input files should be easily adaptable.

The utility works for the `FilNam.in` files that I used. In addition, to ease the usage, an example input file `EPJC_72_2046Fort.in` is provided together with `ForttoBlue.inp`. After creating the function `B_EPJC_72_2046Fort.cxx` with `ForttoBlue()`, the result from the Fortran software on `EPJC_72_2046Fort.in`, as well as those from running the newly created function for `Flag` $= 0$, i.e. `B_EPJC_72_2046Fort(0)` or the distributed example `B_EPJC_72_2046(0)`, are identical.

6 Hints on the software installation

The software release x.y.z is distributed via the corresponding hepforge project page [37] as a gziped tar file named Blue-x.y.z.tar.gz, where the present release is $x.y.z = 2.4.0$. The result of this software is not expected to depend on the actual release of the ROOT package. By now, most tests were performed with the present Pro Release 6.18/04, and the release of the ROOT docker container 6.12/07. Before installation of the BLUE package, the wanted installed release of ROOT needs to be selected. Alternatively, a similar situation is achieved by making use of the ROOT docker container for `Ubuntu 16`.

As an example, using singularity [3] and executing:

1. `singularity -d pull docker://rootproject/root-ubuntu16`
2. `singularity shell root-ubuntu16.simg`

gives access to the ROOT 6.12/07 with `ROOTSYS=/usr/local/`.

The further installation steps are identical. Here they are described for using ROOT 6.18/04 on `lxplus7` at CERN with the bash shell. For this one would `export` the three environment variables:

```
ROOTSYS=/cvmfs/sft.cern.ch/lcg/app/releases/ROOT/6.18.04/x86_64-centos7-gcc48-opt/
```

[3]See: singularity and the Ubuntu singularity-container package.

```
PATH=$PATH:$ROOTSYS/bin
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$ROOTSYS/lib
```

e.g. in the `.bashrc` file and execute:

```
source $ROOTSYS/bin/thisroot.sh
```

from the shell prompt. After this, to install and use the BLUE package perform the following steps:

1. Unzip the file: `gzip -d Blue-2.4.0.tar.gz`
2. Untar the file: `tar -xf Blue-2.4.0.tar` in the directory `2.4.0`
3. Compile the class: `make`
4. Start ROOT: `root`
5. Load the BLUE library: `gSystem->Load("libBlue.so");`
6. Get access to any of the example functions: e.g. `.L B_EPJC_72_2046.cxx++`
7. Execute a specific combination of this example: `B_EPJC_72_2046(1)`

For a more automated usage see the above descriptions of `BlueOne` and `BlueAll`. Finally, using the script `Install` the realease x.y.z can be installed and the examples run by typing `Install Blue-x.y.z.tar.gz`.

In addition to the interface described in this manual, the software contains a number of `private:` member functions. However, differently from regular `C++` code, when the ACLiC system is used for the examples as suggested above, these member functions are **not** prohibited from being used outside of the class. Clearly, using those functions is strongly discouraged and can lead to unexpected results.

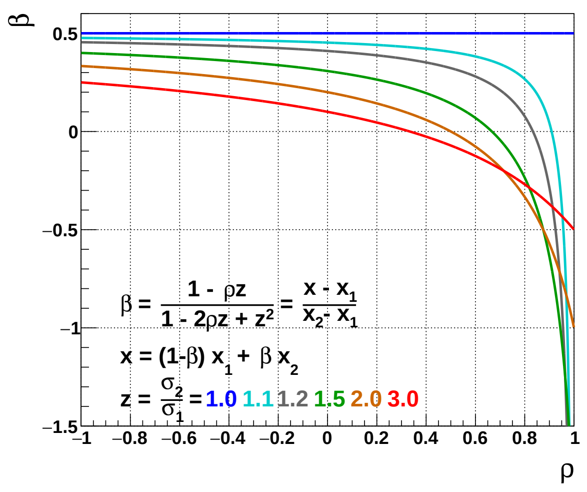
7 Conclusions

In this manual, a software package to perform the combination of several estimates of a number of observables is presented. The software is freely available from the corresponding hepforge project page. Given it is based on ROOT, it is distributed under the GNU Lesser General Public License. When using this software in publications, please give reference to the Software homepage [37] and to Ref. [5]. Should you spot any mistake or peculiarity, please inform the author. If you want to be informed about new software releases by e-mail, let me know by e-mail.

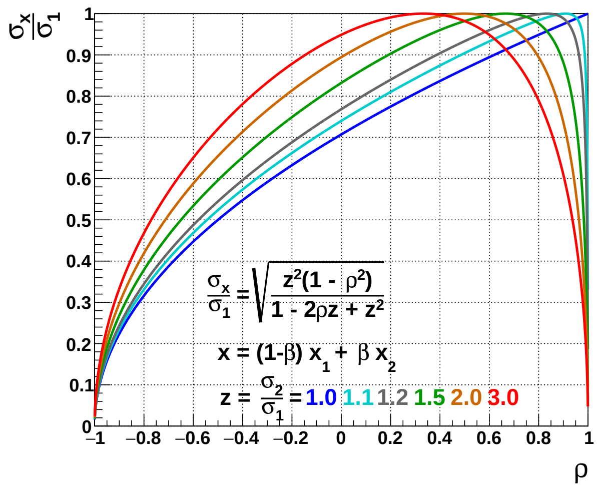
Acknowledgements

I like to acknowledge the many useful discussions on the project I had, and the helpful support on technical issues provided by a number of people. I am very grateful to Sven Menke for his valuable help on a number of implementation issues. I like to warmly thank Giorgio Cortiana for intensively using the code and providing detailed feedback. The assistance with singularity from Daniel Britzger is much appreciated.

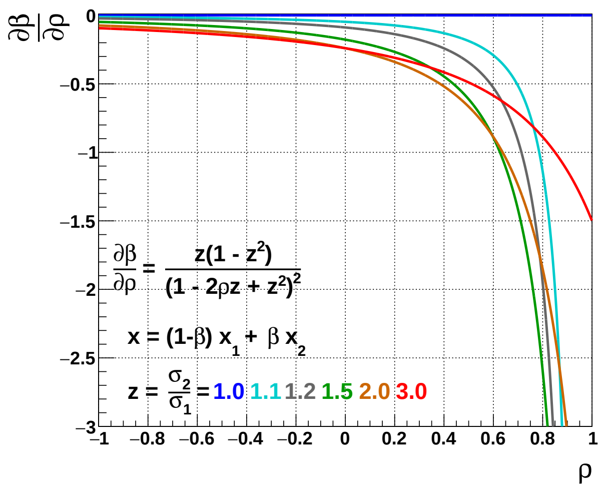
I am also constantly profiting from feedback provided by users of this software. This help in making it better is much appreciated. My thanks go to: Simone Bifani, Thorsten Chwalek, Frederic Deliot, Julien Noce Donini, Marc Escalier, Luca Lista, Mark Owen, Amartya Rej, and Francesco Spano, my apologies go to those I forgot to mention.



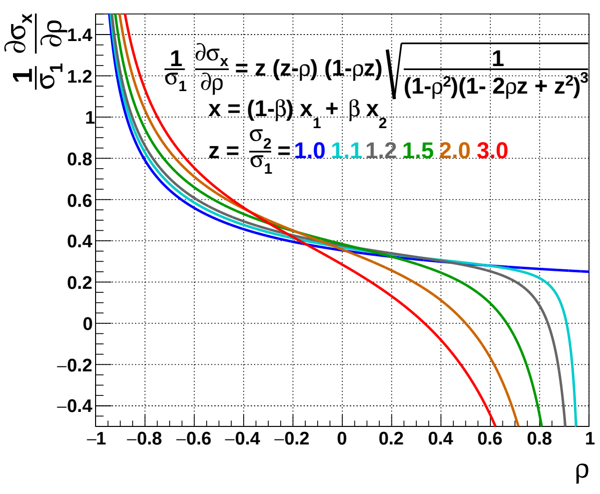
(a) β as a function of ρ



(b) σ_x/σ_1 as a function of ρ

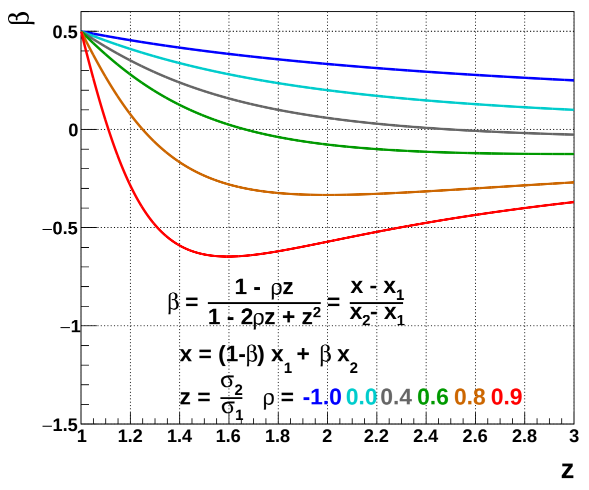


(c) $\partial\beta/\partial\rho$ as a function of ρ

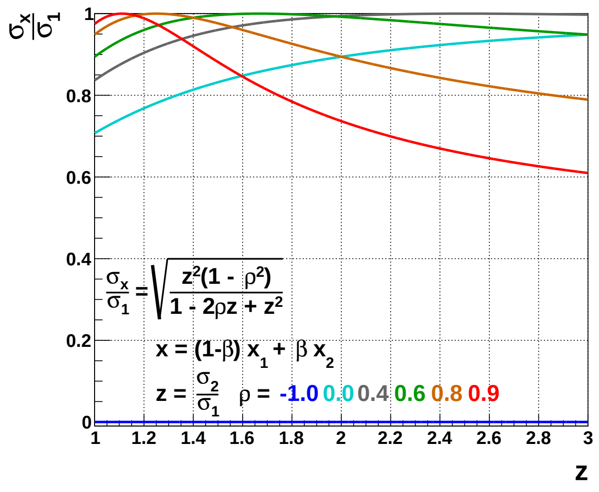


(d) $1/\sigma_1\ \partial\sigma_x/\partial\rho$ as a function of ρ

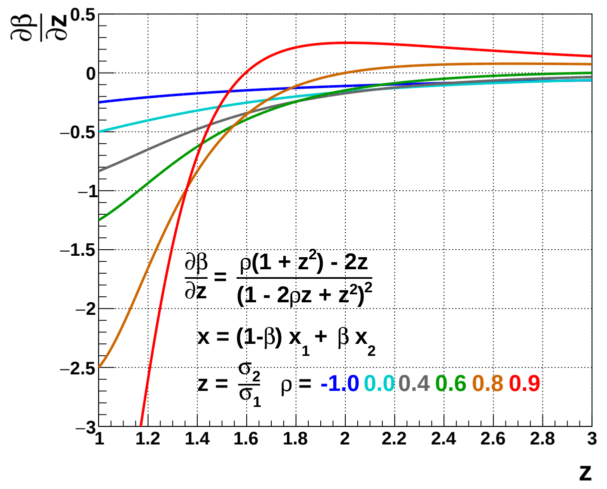
Figure 2: The results of Eqs. 2–5 as functions of ρ for a number of z values. Shown are (a) β and (b) σ_x/σ_1 and their derivatives with respect to ρ, (c) $\partial\beta/\partial\rho$ and (d) $1/\sigma_1\ \partial\sigma_x/\partial\rho$.



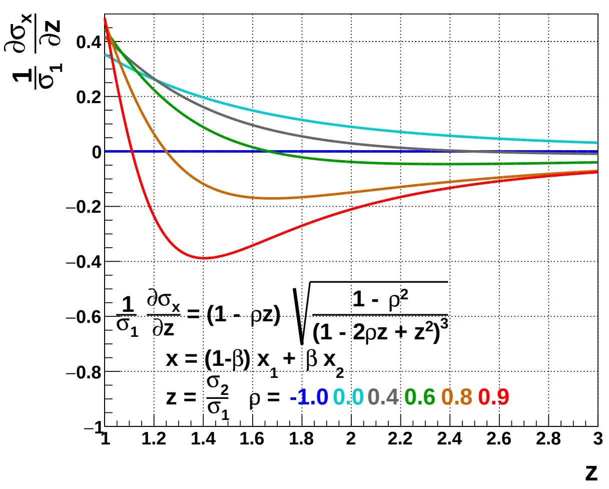
(a) β as a function of z



(b) σ_x/σ_1 as a function of z

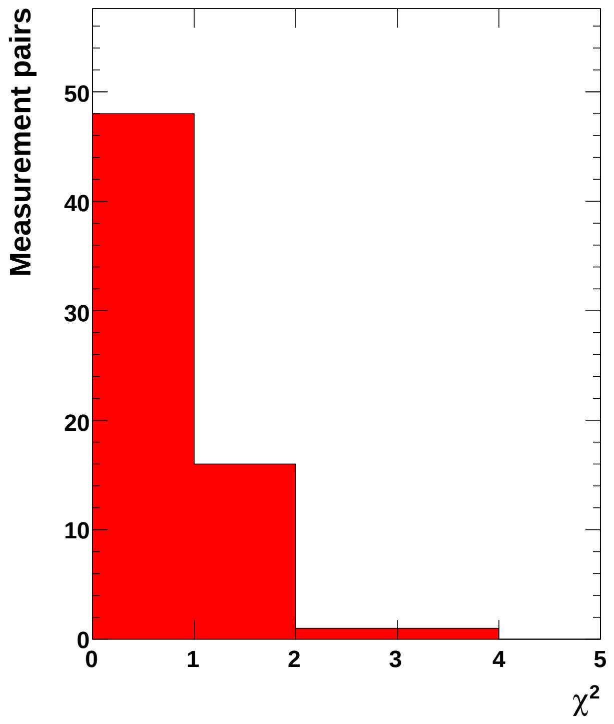


(c) $\partial\beta/\partial z$ as a function of z

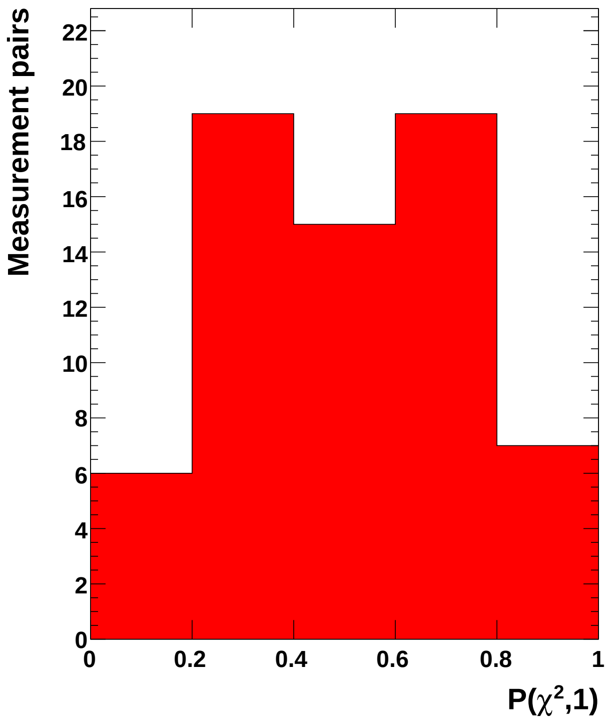


(d) $1/\sigma_1\ \partial\sigma_x/\partial z$ as a function of z

Figure 3: The results of Eqs. 2–3, 6–7 as functions of z for a number of ρ values. Shown are (a) β and (b) σ_x/σ_1 and their derivatives with respect to z, (c) $\partial\beta/\partial z$ and (d) $1/\sigma_1\ \partial\sigma_x/\partial z$.

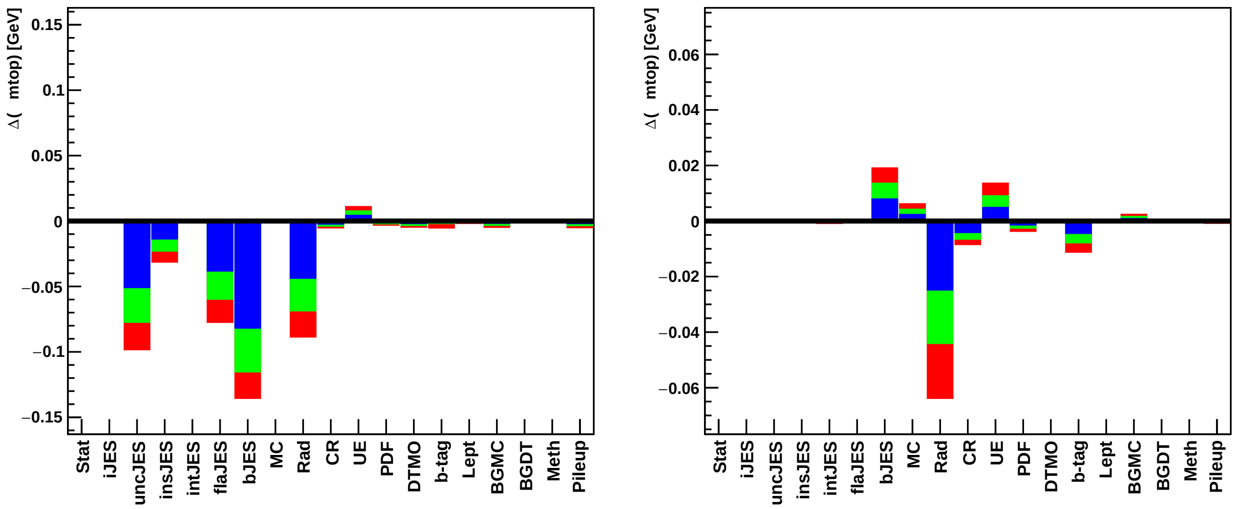


(a) The χ^2 distribution

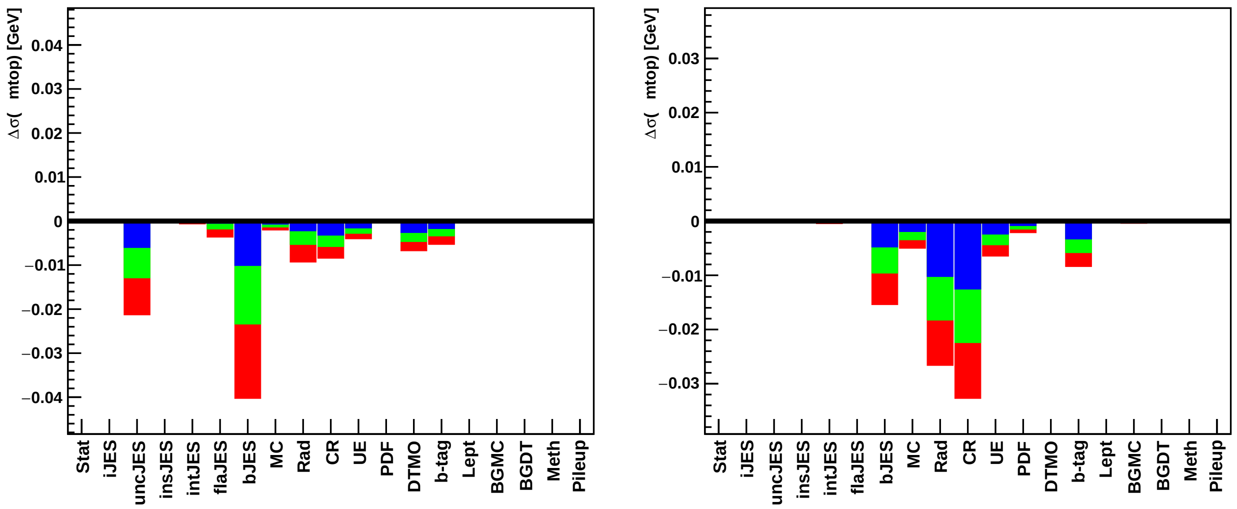


(b) The $P(\chi^2, 1)$ distribution

Figure 4: The results of the compatibility investigation using `CompatEst()` for the example `B_arXiv_1305_3929.cxx(1)`. Shown are (a) the χ^2 distribution, and (b) the corresponding $P(\chi^2, 1)$ distribution for the observable under study.

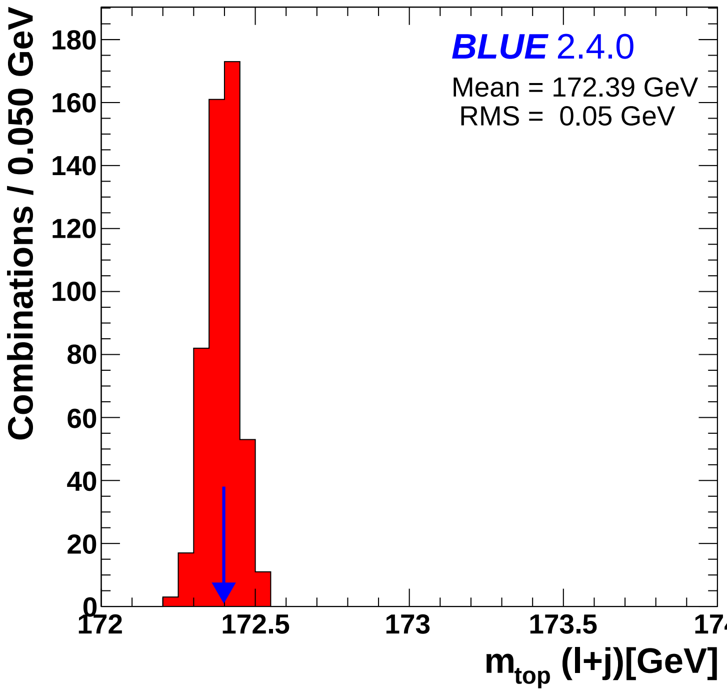


(a) The differences in the values

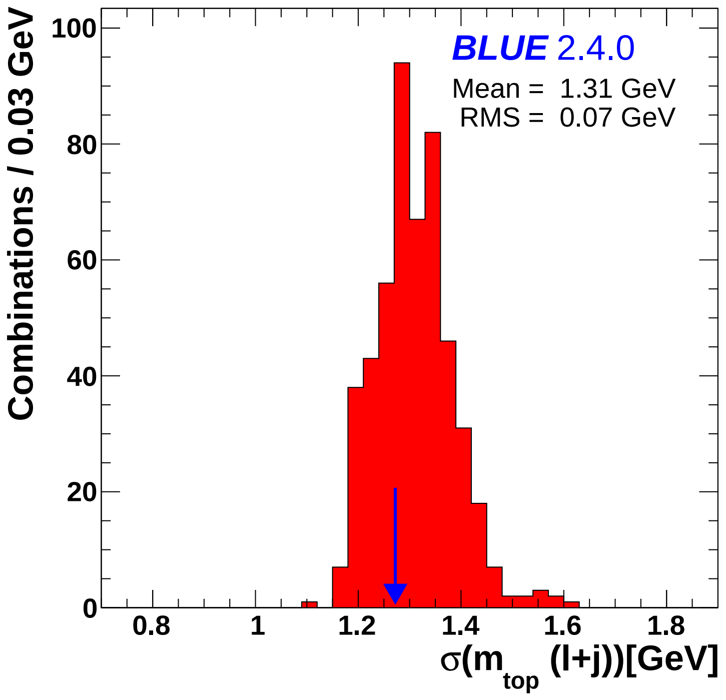


(b) The differences in the uncertainties

Figure 5: The results of the correlation scan of `SolveScaRho()` using `PrintScaRho(FilNam)`, taken from the example `B_ATLAS_CONF_2013_102(9)`. Shown are (a) the observed shifts (actual - default) in the value, and (b) the corresponding shifts in the uncertainty in the combined result for the observable under study and separated into the groups defined in the constructor. The left histograms are for group $\ell = 0$, the right ones for group $\ell = 1$. They show the results at step four (blue), seven (green) and ten (red) of the variation, where the scanned range of the correlation was defined in the call to `SolveScaRho()`.



(a) Distribution of the combined result



(b) Distribution of the uncertainty in the combined result

Figure 6: Examples of results of `SolveScaSta(2)` using `PrintScaSta()` for the scan of the uncertainties using their statistical precision, taken from the example `B_PLB_761_350(2)`. Shown are (a) the distribution of the combined value, and (b) the distribution of the corresponding uncertainty in the combined result.

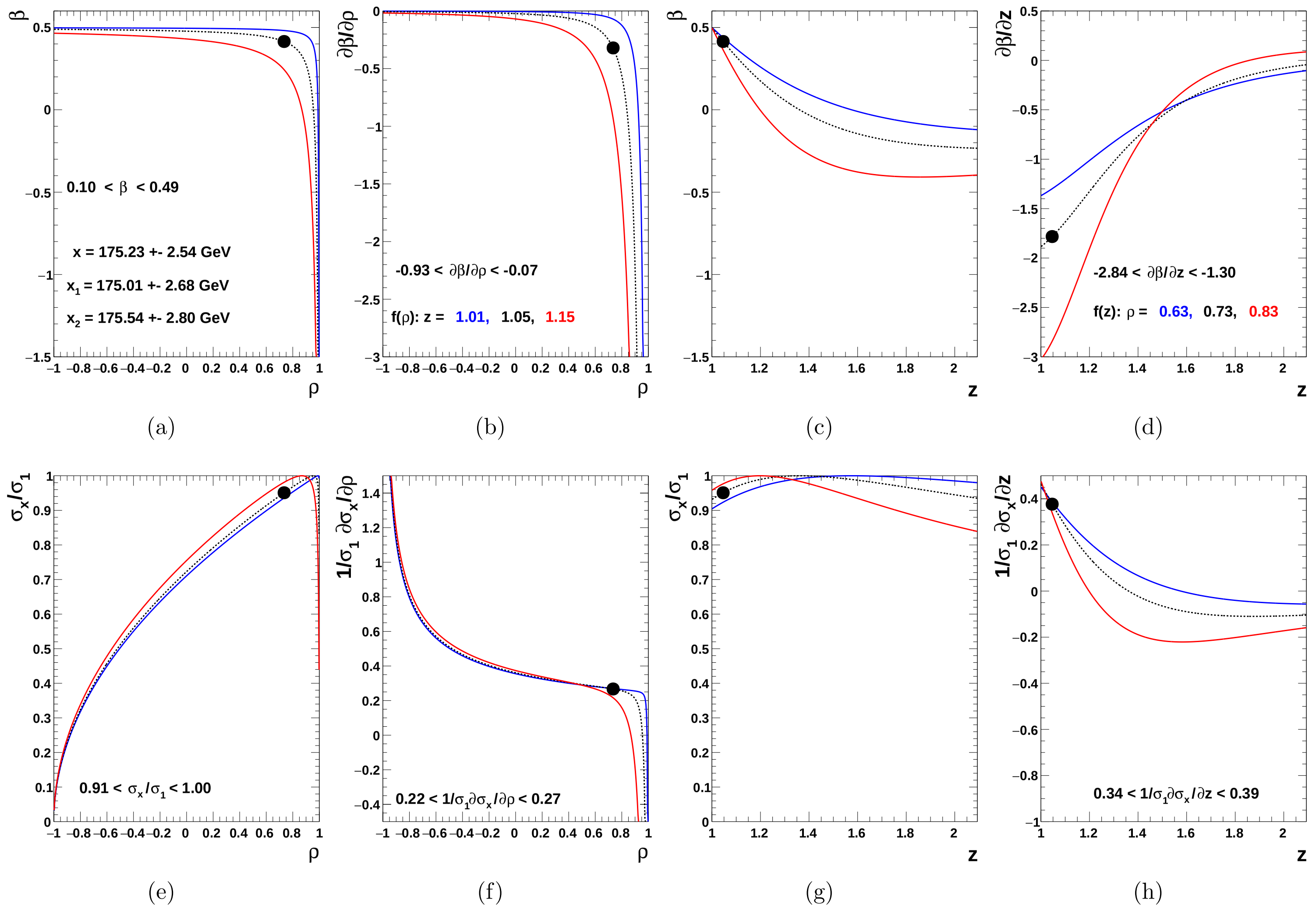


Figure 7: The sub-figures correspond to Figures 2–3 for the example `B_EPJC_72_2046(3)`, i.e. the black point represents the actual values of ρ and z. In (a) also the estimates x_1 and x_2, as well as the combined value x, together with their uncertainties, are listed. In each sub-figure three curves are shown in which, for parameters shown as a function of ρ (or z), the value of z (or ρ) is varied. The curves corresponding to the minimum/central/maximum value of this variation are shown in blue/black/red, and the three values used for z and ρ are given in (b) and (d), respectively. For the derivatives of β and σ_x/σ_1 with respect to ρ and z, for each sub-figure the range of observed parameter values is given. This range is obtained for the three curves shown, while keeping the respective value of the other parameter. As an example in (b) the range in $\partial\beta/\partial\rho$ at $\rho = 0.73$ is quoted observed when changing z from 1.01 to 1.15. Finally, for β and σ_x/σ_1 their full range is quoted in (a) and (e). This range is obtained using all nine possible pairs of the ρ and z values.

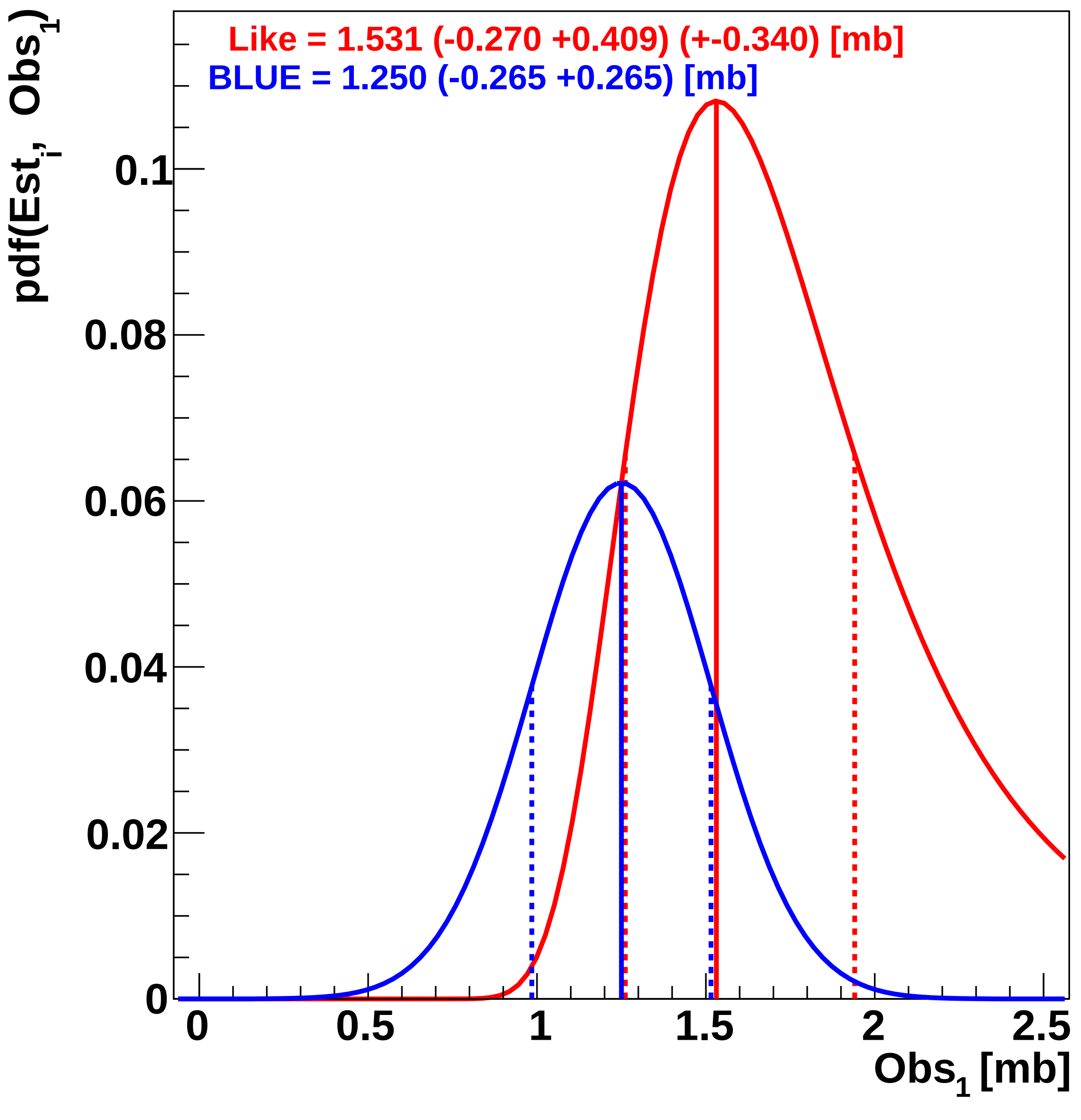
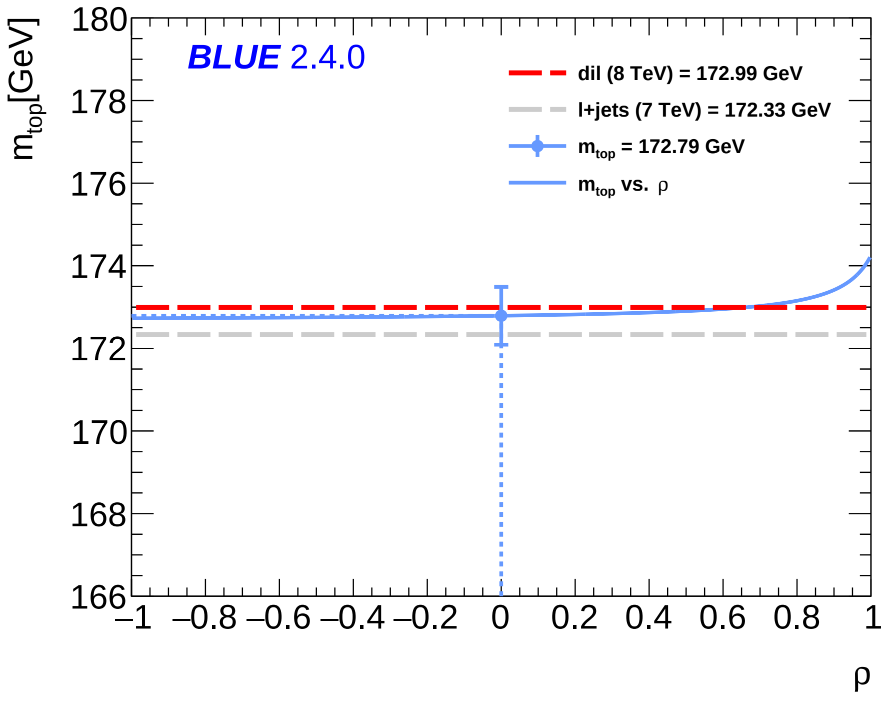
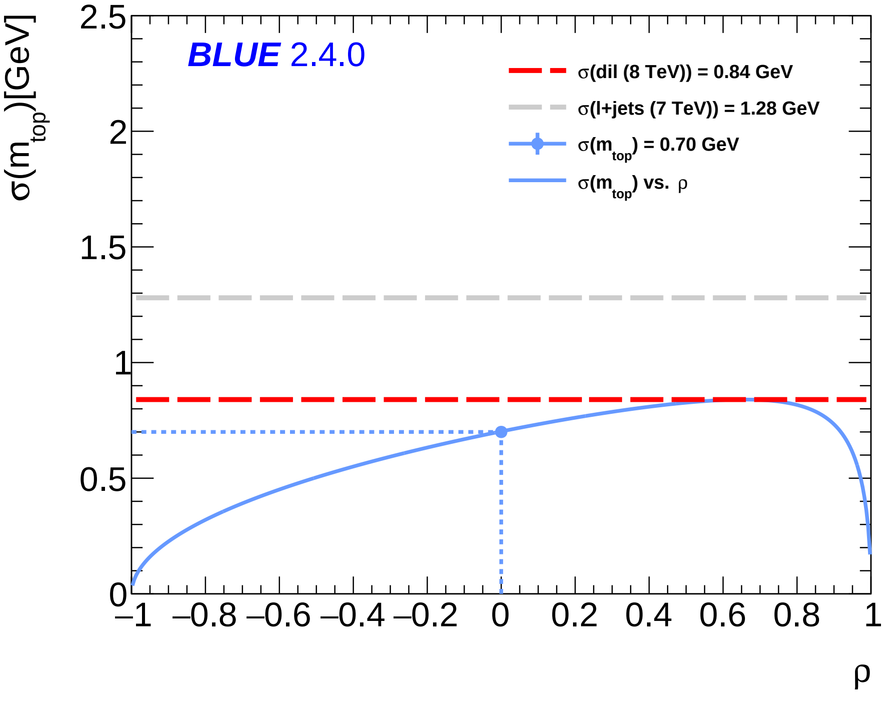


Figure 8: The results of `InspectLike()` taken from the example `B_Peelles(0)`. Shown are the results from the likelihood fit (red) in comparison to the result from the BLUE combination (blue) with relative uncertainties in scenario $\mathcal{A}$ of Table 1 of Ref. [5].



(a) Combined result as a function of ρ



(b) Uncertainty of combined result as a function of ρ

Figure 9: The result of `DisplayPair()` for the combination of two estimates, taken from the example `B_PLB_761_350(0)`. Shown are (a) the individual estimates and their combination, together with the combined result as a function of the correlation. Figure (b) shows similar quantities, but this time for the uncertainties.

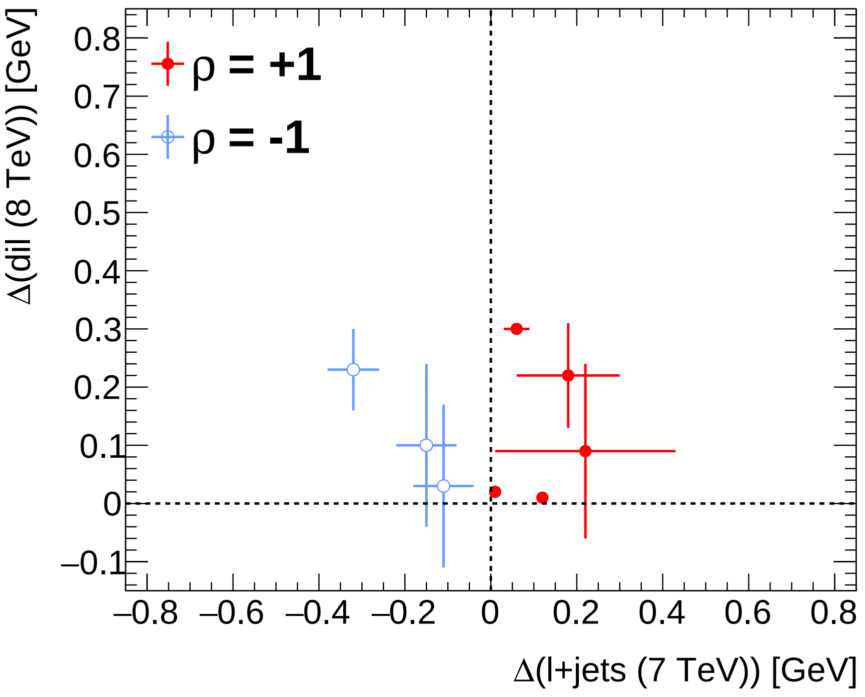


Figure 10: The result of `CorrelPair()` for a pair of estimates, taken from the example `B_PLB_-761_350(0)`. This figure only contains part of the information of Figure 3(b) of Ref. [11], namely all sources of Table 2 that have $\rho\pm1$ are displayed. In contrast, for uncertainty sources composed of subcomponents, e.g. the jet energy scale uncertainty induced uncertainty in the top quark mass, Figure 3(b) of Ref. [11] displays all subcomponents of the eigenvector decomposition. In addition, given for many sources the definition of what is called the + variation is an arbitrary choice, quadrants three and four of the original figure, are projected onto quadrants one and two.

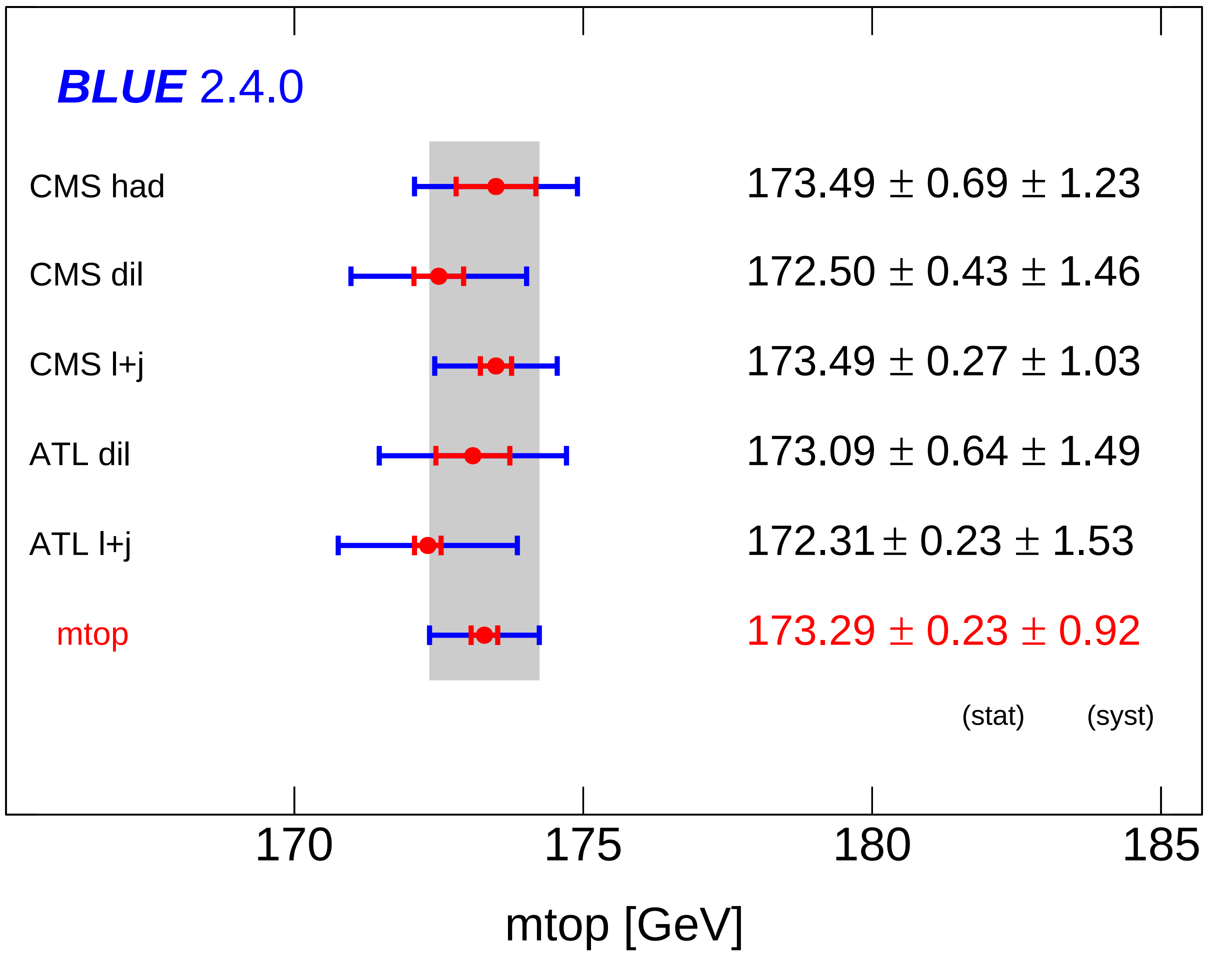


Figure 11: The result of `DisplayResult()` taken from the example `B_ATLAS_CONF_2013_-102(0)`. This figure shows the individual measurements of the top quark mass and the combined result quoted in red.

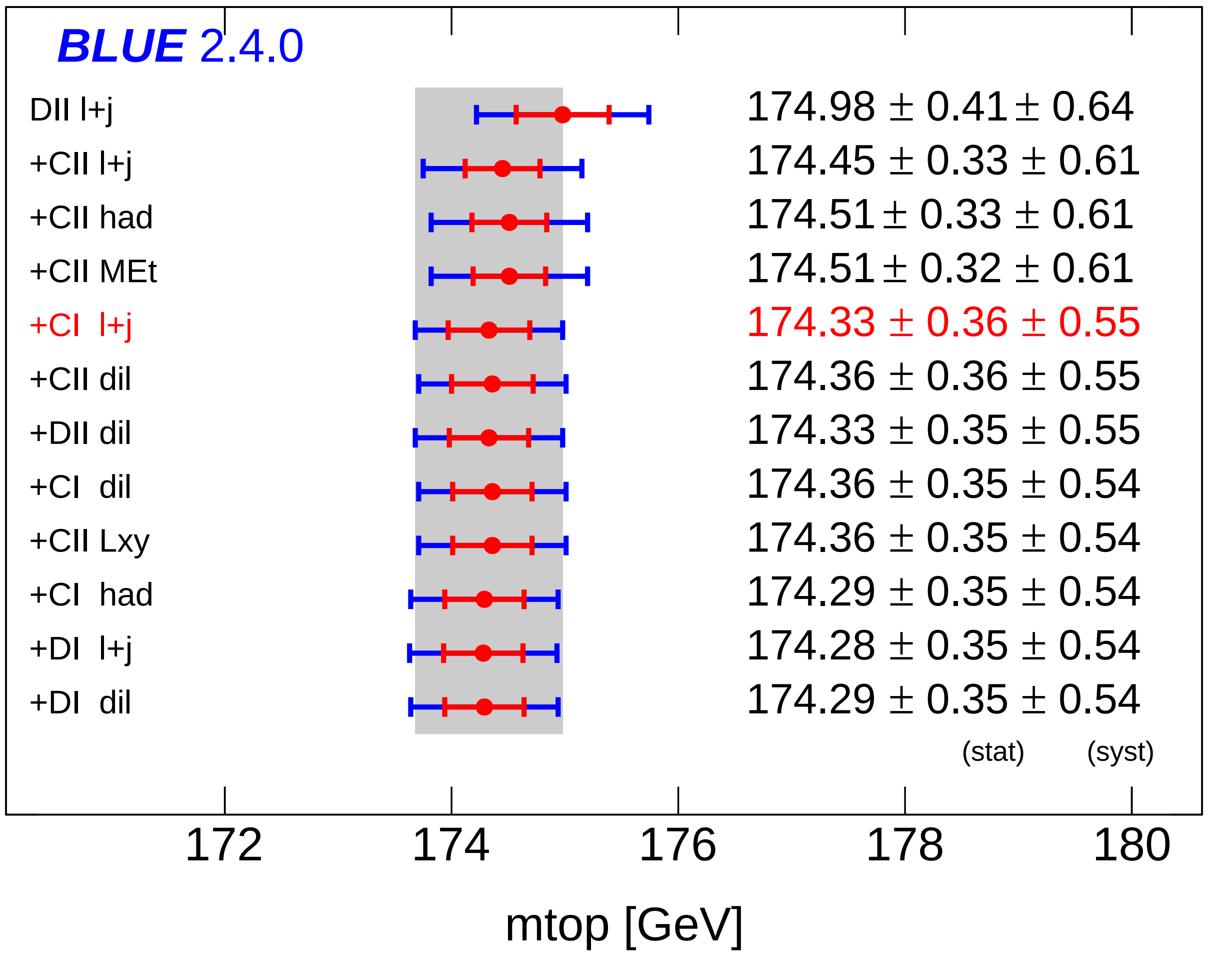


Figure 12: The result of `DisplayAccImp()` taken from the example `B_arXiv_1608_01881(1)`. Each line of this figure shows the combined result, when successively adding one result at a time. All combinations below the one quoted in red never improve the precision by more than 1%.

Index

A Release notes

The latest changes made to the software, restricted to the last three major releases, are listed in reverse order. Only the main points are given, for details of added features the reader is referred to the description of the interface in the main part of the text. Whenever a new release is made all examples are run and it is verified that the differences are solely due to the new features of the package. It would be helpful if users do the same with their `B_name.cxx` routines, and report any peculiarities to the author.

Changes from 2.3.0 to 2.4.0

1. Update the hints on the software installation to also cover the ROOT docker container.
2. Add `GetCompatEst()` and `GetCompatObs()`.
3. Add results of `GetCompatEst()` and `GetCompatObs()` to `LatexResult()`.
4. Add more hints about the usage of the software to the examples. Use `grep Shows B_*cxx | grep ThisFunction` to see examples for how the function `ThisFunction` is used.
5. Add format `ForChi`, adapt `SetFormat()` and `LatexResult()` and a few examples.
6. Expand the section about usage of the software in this manual.

Changes from 2.2.0 to 2.3.0

1. Fix a bug in `LatexResult()` and `CorrelPair()` introduced in 2.2.0, which resulted in wrong correlation matrices been printed (or wrong points being displayed) for the case in which sources of uncertainties were disabled.
2. Now also allow a call for a single estimate of an observable, although nothing can be combined.
3. Add print statement to `PrintScaRho`.
4. Add examples on the LHCb combination of various brancing fractions of D meson decays `B_JHEP_03_176`, and the CMS combination of the $W\gamma\gamma$ and $Z\gamma\gamma$ cross-sections `B_JHEP_-10_072`.

Changes from 2.1.0 to 2.2.0

1. Adapt to a consistent use of `const` in the method calls.
2. Update `PrintResult()`, `LatexResult()` and `DisplayResult()` to allow for variances of individual sources of uncertainty of the observables to be smaller than zero, see Eq. 18 of Ref. [2]. The occurrence of this is indicated by *negative* uncertainties. See description of `PrintResult()`.
3. Expand `PrintMatrix()` to allow for printing a range of rows and columns.
4. Add the possibility to define default formats for print-out of values and a default unit, e.g. GeV in `SetFormat()`. Adapt `PrintInspectLike()`, `PrintScaRho()`, `DrawSens()`, `PrintResult()` and `PrintEst()`, to use those.
5. Add the possibility to keep the user root setup using `SetNoRootSetup()`.
6. Add the possibility to use a logo `SetLogo()`.
7. Change print-out in `GetAccImpIndEst()` and `PrintEst()`.
8. Add several new implementations of `FillEst()` and `FillCor()`.
9. Add `GetCor()` to return a correlation matrix for an individual uncertainty source.

10. Remove a bug in `SolveAccImp()` that was present when using more than one observable. In this case, the combined result for the situation of one estimate for the observable under study, and all estimates of the remaining observables, was equal to the single result. This means the impact of the estimators of the other observable was ignored. Now this is properly taken into account, and the estimators of the other observables are allowed to alter the single estimate combination of the observable under study.
11. Add `SolveScaSta()` for investigating the stability of a result with respect to the statistical precision of the uncertainties. Implemented are: the solving method and the corresponding filling `FillSta()` and printing `PrintScaSta()` methods, as well as getters to retrieve the input `GetSta()` and the result `GetStaRes()`. In addition, `LatexResult()` has been updated to also list the statistical precision if present.
12. Add getters to retrieve the original numbers of estimates `GetNumEst()`, uncertainties `GetNumUnct()` and observables `GetNumObs()`.
13. Add `CorrelPair()` to display the correlation of a pair of estimates for uncertainty sources with $\rho_{ijk} = \pm 1$.
14. Add `DisplayPair()` to display x and σ_x for a pair of estimates.
15. Correct wrong formats of `printf` statements in `PrintAccImp()`, `PrintCor()`, `InspectPair()` and `SolveScaRho()`.
16. Expand the description on the software installation.
17. Add examples on: the ATLAS combinations of the top quark mass `B_EPJC_75_330`, `B_-PLB_761_350`, `B_EPJC_79_290` and of the top quark pole mass `B_EPJC_74_3109`, the CDF simultaneous measurement of the top quark mass in the lepton+jets and dilepton channels `B_PRD79_092005`, the CMS combinations of the top quark mass `B_CMS_PAS_2014_015` and `B_PRD_93_072004`, the Tevatron combination of the top quark mass `B_arXiv_1608_01881`, and the LHC combination of the Higgs boson mass `B_PRL_114_191803`, and an example to show how negative variances occur `B_NegVar`.
18. Correct a bug in `B_ATLAS_CONF_2014_052`.
19. Correct a bug in `B_ATLAS_CONF_2013_033` for Flag == 4.

References

[1] L. Lyons and D. Gibaut and P. Clifford, How to combine correlated estimates of a single physical quantity, Nucl. Instr. and Meth. A 270 (1988) 110.

[2] A. Valassi, Combining correlated measurements of several different quantities, Nucl. Instr. and Meth. A 500 (2003) 391.

[3] M. Grunewald, private communication, unpublished software.

[4] R. Brun and F. Rademakers, ROOT - An Object Oriented Data Analysis Framework, Nucl. Instr. and Meth. A 389 (1997) 81, Proceedings of AIHENP'96 Workshop, Lausanne, Sep. 1996.
URL `https://root.cern.ch`

[5] R. Nisius, On the combination of correlated estimates of a physics observable, Eur. Phys. J. C74 (2014) 3004. `arXiv:arxiv:1402.4016`.

[6] ATLAS Collaboration, Measurement of the top quark mass in the $t\bar{t} \rightarrow$ lepton+jets and $t\bar{t} \rightarrow$ dilepton channels using $\sqrt{s} = 7\,\text{TeV}$ ATLAS data, Eur. Phys. J. C 75 (2015) 330. `arXiv:1503.05427`, `doi:10.1140/epjc/s10052-015-3544-0`.

[7] L. Lyons and A.J. Martin and D.H. Saxon, On the determination of the B lifetime by combining the results of different experiments, Phys. Rev. D 41 (1990) 982.

[8] A. Valassi and R. Chierici, Information and treatment of unknown correlations in the combinination of measurements using the BLUE method (v3). `arXiv:1307.4003`.

[9] ATLAS Collaboration, Measurement of the top quark mass with the template method in the $t\bar{t} \rightarrow$ lepton + jets channel using ATLAS data, Eur. Phys. J. C 72 (2012) 2046. `arXiv:1203.5755`, `doi:10.1140/epjc/s10052-012-2046-6`.

[10] R.J. Barlow, Statistics: a guide to the use of statistical methods in physical sciences, John Wiley & Sons Ltd., 1989, ISBN 0 471 92295 1.

[11] ATLAS Collaboration, Measurement of the top quark mass in the $t\bar{t} \rightarrow$ dilepton channel from $\sqrt{s} = 8\,\text{TeV}$ ATLAS data, Phys. Lett. B 761 (2016) 350. `arXiv:1606.02179`, `doi:10.1016/j.physletb.2016.08.042`.

[12] The Tevatron Electroweak Working Group for the CDF and DØ Collaborations, Combination of CDF and DØ results on the mass of the top quark using up to 5.8 fb^{-1} of data (v3). `arXiv:1107.5255`.

[13] The Tevatron Electroweak Working Group for the CDF and DØ Collaborations, Combination of CDF and DØ results on the mass of the top quark using up to 8.7 fb^{-1} at the Tevatron (v2). `arXiv:1305.3929`.

[14] ATLAS, CDF, CMS and DO Collaborations, First combination of Tevatron and LHC measurements of the top-quark mass, ATLAS-CONF-2014-008 (2014).
URL `https://cds.cern.ch/record/1669819`

[15] The Tevatron Electroweak Working Group for the CDF and DØ Collaborations, Combination of CDF and DØ results on the mass of the top quark using up to 9.7 fb^{-1} at the Tevatron (v1). arXiv:1407.2682.

[16] The Tevatron Electroweak Working Group for the CDF and DØ Collaborations, Combination of CDF and DØ results on the mass of the top quark using up to 9.7 fb^{-1} at the TevatronarXiv:1608.01881.

[17] ATLAS and CMS Collaborations, Combination of ATLAS and CMS results on the mass of the top quark using up to 4.9 fb^{-1} of data, ATLAS-CONF-2012-095 (2012).
URL https://cds.cern.ch/record/1460441

[18] ATLAS and CMS Collaborations, Combination of ATLAS and CMS top-quark pair cross-section measurements using proton–proton collisions at $\sqrt{s} = 7$ TeV, ATLAS-CONF-2012-134 (2012).
URL https://cds.cern.ch/record/1478422

[19] ATLAS and CMS Collaborations, Combination of the ATLAS and CMS measurements of the W-boson polarization in top-quark decays, ATLAS-CONF-2013-033 (2013).
URL https://cds.cern.ch/record/1527531

[20] ATLAS and CMS Collaborations, Combination of single top-quark cross-sections measurements in the t-channel at $\sqrt{s} = 8$ TeV with the ATLAS and CMS experiments, ATLAS-CONF-2013-098 (2013).
URL https://cds.cern.ch/record/1601029

[21] ATLAS and CMS Collaborations, Combination of ATLAS and CMS results on the mass of the top-quark using up to 4.9 fb^{-1} of $\sqrt{s} = 7$ TeV LHC data, ATLAS-CONF-2013-102 (2013).
URL https://cds.cern.ch/record/1601811

[22] ATLAS and CMS Collaborations, Combination of ATLAS and CMS $t\bar{t}$ charge asymmetry measurements using LHC proton–proton collisions at $\sqrt{s} = 7$ TeV, ATLAS-CONF-2014-012 (2014).
URL https://cds.cern.ch/record/1670535

[23] ATLAS and CMS Collaborations, Combination of cross-section measurements for associated production of a single top-quark and a W boson at $\sqrt{s} = 8$ TeV with the ATLAS and CMS experiments, ATLAS-CONF-2014-052 (2014).
URL https://cds.cern.ch/record/1951032

[24] ATLAS and CMS Collaborations, Combination of ATLAS and CMS top quark pair cross section measurements in the $e\mu$ final state using proton–proton collisions at $\sqrt{s} = 8$ TeV, ATLAS-CONF-2014-054 (2014).
URL https://cds.cern.ch/record/1951322

[25] CMS Collaboration, Combination of the CMS top-quark mass measurements from Run 1 of the LHC, CMS-PAS-TOP-14-015.
URL https://cds.cern.ch/record/1951019

[26] ATLAS Collaboration, Measurement of the $t\bar{t}$ production cross-section using $e\mu$ events with b-tagged jets in pp collisions at $\sqrt{s} = 7$ and $8\,\text{TeV}$ with the ATLAS detector, Eur. Phys. J. C 74 (2014) 3109. `arXiv:1406.5375`, `doi:10.1140/epjc/s10052-016-4501-2`.

[27] ATLAS Collaboration, Measurement of the top quark mass in the $t\bar{t} \to$ lepton+jets channel from $\sqrt{s} = 8\,\text{TeV}$ ATLAS data and combination with previous results, Eur. Phys. J. C 79 (2019) 290. `arXiv:1810.01772`, `doi:10.1140/epjc/s10052-019-6757-9`.

[28] LHCb Collaboration, Measurement of the branching fractions of the decays $D^+ \to K^- K^+ K^+$, $D^+ \to \pi^- \pi^+ K^+$ and $D^+_s \to \pi^- K^+ K^+$, JHEP 03 (2019) 176. `arXiv:1810.03138`, `doi:10.1007/JHEP03(2019)176`.

[29] CMS Collaboration, Measurements of the $pp \to W\gamma\gamma$ and $pp \to Z\gamma\gamma$ cross sections and limits on anomalous quartic gauge couplings at $\sqrt{s} = 8\,\text{TeV}$, JHEP 10 (2017) 072. `arXiv:1704.00366`, `doi:10.1007/JHEP10(2017)072`.

[30] R.W. Peelle, Peelle's Pertinent Puzzle, Internal Memorandum, Oak Ridge National Laboratory, Washington DC, USA, unpublished.

[31] S. Chiba and D.L. Smith, A suggested procedure for resolving an anomaly in least-squares data analysis known as 'Peelle‘s Pertinent Puzzle' and the general implications for nuclear data evaluation, ANL/NDM-121.
URL `http://www.osti.gov/scitech/biblio/10121367`

[32] ATLAS Collaboration, Measurement of the Higgs boson mass in the $H \to ZZ^* \to 4\ell$ and $H \to \gamma\gamma$ channels with $\sqrt{s} = 13\,\text{TeV}$ pp collisions using the ATLAS detector, Phys. Lett. B 784 (2018) 345. `arXiv:1806.00242`, `doi:10.1016/j.physletb.2018.07.050`.

[33] CDF Collaboration, First simultaneous measurement of the top quark mass in the lepton + jets and dilepton channels at CDF, Phys. Rev. D 79 (2009) 092005. `arXiv:arXiv:0809.4808`.

[34] The DØ and CDF Collaborations, T. Aaltonen, et al., Combination of CDF and DØW-Boson Mass Measurements, Phys. Rev. D88 (2013) 052018.

[35] CMS Collaboration, Measurement of the top quark mass using proton-proton data at $\sqrt{s} =$ 7 and 8 TeV, Phys. Rev. D 93 (2015) 072004. `arXiv:arXiv:1509.04044`.

[36] ATLAS and CMS Collaborations, Combined Measurement of the Higgs Boson Mass in pp Collisions at $\sqrt{s} = 7$ and $8\,\text{TeV}$ with the ATLAS and CMS Experiments, Phys. Rev. Lett. 114 (2015) 191803. `arXiv:1503.07589`, `doi:10.1103/PhysRevLett.114.191803`.

[37] R. Nisius, BLUE: A ROOT class to combine a number of correlated estimates of one or more observables using the Best Linear Unbiased Estimate method.
URL `http://blue.hepforge.org`